ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

DETECCIÓN DE LUNARES EN LA PIEL MEDIANTE PROCESADO DIGITAL DE IMAGEN

Jorge Sarrías Erdozain

Armando Malanda Trigueros

Pamplona, 11-Septiembre-2012

ÍNDICE

1-INTRODUCCIÓN

El profesor Armando Malanda Trigueros del Departamento de Ingeniería Eléctrica y Electrónica de la Universidad Pública de Navarra solicitó a Jorge Sarrías Erdozain, estudiante de Ingeniería Técnica de Telecomunicación especialidad Sonido e Imagen el desarrollo de un proyecto de procesado digital de imagen para la detección de lunares en imágenes de la piel.

La propuesta trata de diseñar un programa que permita a pacientes de dermatología llevar un chequeo rutinario de manera sencilla y sin salir de sus propias casas. Mediante una imagen digital de parte de la piel, el programa deberá ser capaz de marcar los lunares y marcas significativas, para después analizarlos y compararlos con imágenes anteriores. El programa enumerará las marcas y mostrará al paciente los cambios a modo de prevención.

Este proyecto se encargará de la primera parte, marcar los lunares y marcas significativas. El camino a seguir después será el de enumeración de estas marcas y posterior análisis. Para ello se han diseñado e implementado una serie de algoritmos basados en las técnicas de procesado digital de imagen aprendidas en las asignaturas de Procesado Digital de Imagen I y Procesado Digital de Imagen II cursadas durante la carrera.

A grandes rasgos el proyecto consistirá en la creación de un programa en Matlab, que tendrá por entrada imágenes de diferentes zonas del cuerpo de una persona y dará como salida la misma imagen con los lunares y marcas resaltados con un color llamativo. En los siguientes apartados se explicarán el material y conocimientos necesarios para la realización de los programas, así como otros datos de interés.

2-OBJETIVOS

- Diseño de un algoritmo para la detección de marcas sobre imágenes de la piel (lunares, pecas,…). Teniendo en cuenta características que los diferencien de la piel y utilizando técnicas de procesado digital de imagen.
- Implementación del algoritmo mediante el programa Matlab, de diseño matemático.
- Prueba de los programas y análisis de los resultados, realizando un estudio que nos indique si el programa es bueno o si hay que perfeccionarlo.
- Mejora del programa a partir del análisis de los resultados obtenidos, intentando perfeccionar problemas que pueden dar el vello, textura de la piel, otros bordes no correspondientes a lunares,…
- Asentar las bases para poder ampliar el algoritmo permitiendo que indique al usuario la aparición de nuevas marcas y variaciones en la piel para prevención.

3-PROBLEMÁTICA

A priori el problema parece tener fácil solución: los lunares son de diferente color que el resto de la imagen, se realiza un barrido de la imagen y se marcan los píxels que cumplan esas características de color. El problema radica en que colores de un lunar pueden repetirse en la piel de otras imágenes o incluso en una misma, en una zona más alejada y que esté peor iluminada, produciéndose la señalización de zonas que no corresponden a marcas.

Sin embargo, los lunares cumplen una propiedad a tener en cuenta para diferenciarlos: son de color diferente a la zona de piel que las rodea. Podemos aplicar una de las técnicas de procesado de imagen: la detección de bordes. Recordemos que lo bordes son zonas frontera que separan regiones de la imagen con características de color diferentes. Las técnicas utilizadas para la detección de bordes son los filtros realzantes (implementan la primera derivada espacial) o el Laplaciano (implementa la segunda derivada espacial). El Laplaciano es muy sensible al ruido, produce bordes dobles y no detecta la dirección de los bordes, así que lo desechamos directamente. Se han considerado tres filtros realzantes: High-Boost, Roberts, Prewitt y Sobel. El elegido fue el Prewitt que empíricamente resultó el más adecuado porque los demás mostraban detalles excesivos confundiendo marcas de la piel con la propia textura de ésta.

Tras este procesado, todavía nos encontramos con más problemas: hay bordes que no corresponden a lunares (bordes producidos por el vello, axilas, hombros,…), se señalan los límites de la imagen,… Por ello se decidió realizar un segundo procesado que, mediante el estudio de color y forma de las marcas, eliminara las que no pertenecían a lunares. Pese a ello, comprobamos que al utilizar algunos de estos filtros, perdíamos marcas que sí eran correctas (por ejemplo, al utilizar un filtro para el vello eliminábamos pecas próximas entre si). Por eso se decidió utilizar tres métodos diferentes, uno con filtro para el vello (filtro P), otro con filtro para bordes (filtro B) y otro sin utilizar ningún filtro.

4-MATERIAL NECESARIO

En la siguiente sección se detalla el material que se ha utilizado para la realización del proyecto además del material necesario para su ejecución. Básicamente se necesita un PC con el software y potencia adecuados equipado también con algún dispositivo de entrada de imágenes (escáner, cámara de fotos digital,…) y algún otro para el almacenamiento de las imágenes (disco duro, grabadora de CD,…).

4.1-CARACTERÍSTICAS DEL PC

Para la ejecución del programa se necesita al menos un PC con procesador Pentium III a 800MHz o similar. Serán necesarios al menos 128MB de memoria RAM. El programa necesitará tan sólo 200KB de espacio en el disco duro aunque se necesitarán más para el almacén de imágenes y el espacio que el propio Matlab ocupa. El PC utilizado para el desarrollo del programa cuenta con un procesador Pentium IV de 2,4 GHz. Dispone de una memoria RAM de 256 MB y un disco duro de 30 GB. Si el PC no cuenta con estos requisitos mínimos, la ejecución del programa puede tornarse excesivamente lenta llegando a congelar el ordenador. El algoritmo exige gran carga computacional y el tiempo de funcionamiento para un equipo de estas características se encuentra en torno a los 10 minutos, para una imagen de aproximadamente 1MB de tamaño y con multitud de lunares. El programa funciona de manera óptima y se reducen drásticamente los tiempos de procesado en equipos con procesadores de más de un núcleo y con memoria RAM de más de 1GB.

4.2-SOFTWARE

Para la programación y ejecución del programa, es necesario tener instalado en el PC el programa Matlab. La versión utilizada en el desarrollo ha sido el Matlab 6.5 incluyendo su accesorio, GUIDE necesario para la creación de interfaces gráficas. Matlab es un potente programa matemático que trata las imágenes como matrices. Permite crear funciones a partir de otras funciones con las que ya cuenta. El algoritmo está formado con varias funciones guardadas en archivos con extensión .m a los que se puede llamar desde Matlab para ver su utilidad por separado. En secciones posteriores se mostrará como se ha trabajado con Matlab y cómo funcionan cada una de las funciones que componen nuestro algoritmo. También se ha utilizado software específico de tratamiento de imágenes: Photoshop CS 8.0.1. Este software ha sido utilizado para comprobar efectos de filtros sobre imágenes, antes de implementarlos en Matlab. También es de utilidad para conocer las componentes RGB o HSV de un píxel mediante la herramienta cuentagotas. Además se ha utilizado para generar imágenes de características determinadas para comprobar el correcto funcionamiento de las diferentes funciones diseñadas. Por último, han sido necesarios los programas incluidos en el paquete Office: Excel y Word, para hacer estudio de colores/ formas de lunares y para la redacción de informes respectivamente.

4.3-IMÁGENES Y CÁMARA UTILIZADA

Será necesario algún dispositivo para la entrada de imágenes al PC. Puede utilizarse escáneres o la propia red de Internet para la adquisición de imágenes, aunque deberán tener una calidad mínima para que el programa funcione correctamente. Para la adquisición de imágenes se ha utilizado en nuestro caso una cámara DSC-P100 de Sony. Las imágenes realizadas tienen una resolución de 640x480 en formato .bmp, ocupando cada archivo unos 900KB.Las imágenes fueron tomadas en una habitación con bastante luz. Se realizaron en mes de julio hacia las 15:20 horas, por tanto la iluminación de la habitación era excelente. En la mayoría de imágenes se incluye una moneda de un euro para tener una idea del tamaño de las marcas y de la imagen en cuestión. Se realizaron 7 fotografías de diferentes partes del cuerpo, aunque la mayoría pertenecen a la espalda. Se buscó un voluntario con multitud de lunares de diferentes tamaños para enfrentarnos con el mayor número de dificultades posible.

5-PROGRAMAS PRECEDENTES

Los programas consultados que utilizan el procesado digital de imagen para la detección de marcas en la piel, tienen como objetivo el diagnóstico de melanomas, es decir, buscan analizar imágenes de la piel para mostrar si un lunar puede ser un melanoma atendiendo a sus características de color y forma.

El ejemplo que más se asemeja a nuestro objetivo, es el de "Análisis de imagen aplicada al diagnóstico de melanomas" realizado por la Universidad Politécnica de Valencia. El método utilizado es similar al caso que se va a proceder a explicar: paso de la imagen a blanco y negro, selección de un umbral, filtrado, cálculo de tamaños y cálculo de colores para el diagnóstico. En el mismo proyecto se detallan problemas similares a los encontrados en nuestro caso: variaciones de color de la piel, problemas con el vello, poros, textura de la piel...Después utilizará un sistema para el diagnóstico de las marcas, clasificándolos en posibles melanomas o no según características de color forma, etc. Sin embargo, es necesario recortar manualmente la zona circundante a la marca, sólo se procesa la marca más grande presente en esta zona y por último y más importante, no se ha realizado la implementación en Matlab.

Dentro del ámbito de la medicina preventiva, este ejemplo no nos resultaría útil ya que no permite mostrar cambios de los lunares con el paso del tiempo, además de que detectará una marca como posible melanoma, cuando se haya desarrollado como tal.

Se ha buscado algún otro programa que sí que permita la prevención, pero no se ha dado con ninguno. Dentro de la medicina preventiva sí que se ha encontrado el programa "Molemap", en el que se utilizan imágenes de alta intensidad que permiten penetrar en la primera capa de la piel mostrando marcas que a simple vista no se verían. Especialistas dermatólogos realizarán un seguimiento de estas imágenes para realizar un diagnóstico preventivo en el caso de que fuera necesario.

Nuestro objetivo sin embargo, es que el paciente mismo pueda realizarse las imágenes (sin precisar de una cámara y una iluminación especiales) y que los cambios no sean detectados por un dermatólogo sino por el propio programa.

6-CONOCIMIENTOS PREVIOS

Para el diseño, implementación y comprensión del algoritmo será necesario tener conocimientos de procesado digital de imagen y de programación. Las técnicas de procesado digital de imagen conllevan filtrado, umbralización, segmentación, morfología matemática,… Estas técnicas serán tratadas en el capítulo correspondiente a la teoría del procesado digital de imagen.

Se necesitarán conocimientos de Matlab tales como llamada a funciones, definición de matrices y variables,… También serán necesarios conocimientos de programación general como bucles, condiciones, definición de variables,… En capítulos posteriores se explicará la implementación de las diferentes funciones mediante estas técnicas de programación y otras funciones presentes en el programa Matlab.

7-ETAPAS DE DESARROLLO

El proyecto ha sido realizado siguiendo una serie de pautas o etapas de desarrollo. El progreso de este proyecto no es lineal, es decir, no hace falta terminar una de las fases anteriores para empezar a realizar una posterior. Además, dado que tiene un alto componente de programación, en muchas ocasiones se tendrá que corregir programas tras hacer comprobaciones sobre las imágenes y ver que los resultados no son los esperados. Pese a esto se enumerarán una serie de fases que se deben completar para el desarrollo del proyecto.

1-RECOGER INFORMACIÓN SOBRE LAS MARCAS: Para poder detectar un lunar, en primer lugar es necesario conocer sus características de forma, color,... que la diferencien del resto de la piel.

2-OBTENER UNA BASE DE DATOS CON IMÁGENES: Ha sido necesario conseguir varias imágenes, que se utilizarán para probar los programas y para ver en qué porcentaje de imágenes funciona, cuántos lunares se pierden, etc.

3-REPASAR LAS TÉCNICAS DE PROCESADO DE IMAGEN: Ha sido necesario hacer un repaso de las técnicas aprendidas durante la carrera y ver cuáles son las necesarias para nuestro objetivo. Estas técnicas engloban desde transformar una imagen en color blanco y negro hasta la implementación de filtros, pasando por histogramas, morfología matemática, etc.

4-IMPLEMENTACIÓN DE LAS TÉCNICAS EN MATLAB: Las técnicas ya mencionadas han sido implementadas en Matlab. Para ello de una forma global se ha necesitado recorrer la imagen píxel por píxel (con un bucle que recorra las filas anidado con otro bucle que recorra las columnas) aplicando las fórmulas matemáticas para cada técnica en cuestión.

5-PROGRAMACIÓN DEL ALGORITOMO GENERAL: Ha sido necesario generar un programa ‘general’ en el que se apliquen las diferentes técnicas ordenadamente y con unos parámetros que han sido estudiados y comprobados para obtener resultados óptimos. Este programa generará 2 imágenes entre las que tendremos que elegir la más adecuada.

6-PROGRAMACIÓN DE UN POSTPROCESADO: Para tener mejores resultados y hacer que el programa funcione con mayor rapidez, se ha creado un postprocesado, en el cual, se introduce una de las imágenes anteriormente seleccionadas dando a la salida otra imagen optimizada.

7-PROGRAMACIÓN DE UN INTERFAZ GRÁFICO: Gracias a la herramienta 'GUIDE' disponible en MATLAB, se ha programado un interfaz gráfico para que la utilización del programa final sea más sencilla y amena.

8-COMPROBACIÓN DE LOS PROGRAMAS: Se ha tenido que comprobar si los programas tienen errores, son correctos, se pueden optimizar,... y si es necesario volver a programarlos.

9-ESTUDIO DE RESULTADOS Y REDACCIÓN DEL INFORME: Se ha comprobado en qué facetas falla el programa, para qué imágenes los resultados son óptimos y para cuáles son malos. Finalmente se ha redactado un informe, donde se explica en detalle las tareas y los resultados obtenidos.

En el siguiente esquema se resume el método utilizado a la hora de realizar el proyecto:

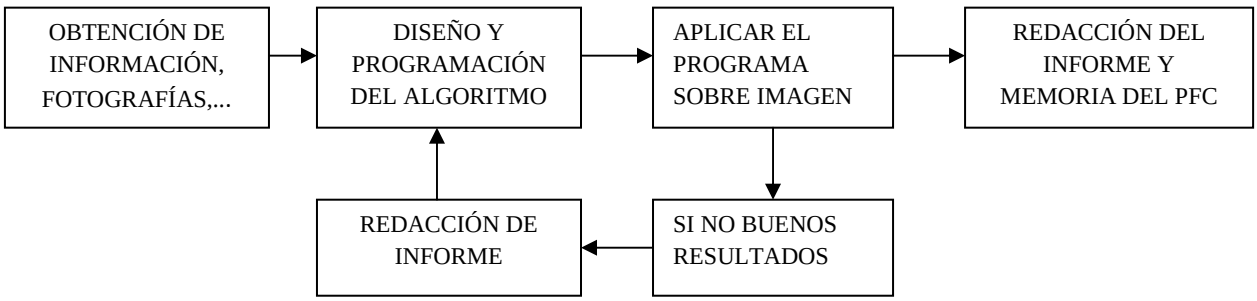


Fig. 7.a: Etapas de desarrollo.

8-EJEMPLO DE FUNCIONAMIENTO

Empezaremos aplicando el programa en dos imágenes a modo de ejemplo, para tener una visión global del funcionamiento del programa. Después podremos explicar con más detalle cada uno de los algoritmos que hemos tenido que implementar para llegar a los resultados deseados.

El programa diseñado se divide en dos partes, la primera se limita a marcar utilizando filtrados Prewitt en diferentes direcciones. La segunda parte eliminará marcas erróneas producidas en el primer procesado, atendiendo a características de los lunares como son forma, color, tamaño, etc. Utilizaremos una imagen sencilla para explicar el primer procesado (ya que en imágenes con muchos bordes apenas dejarían ver los resultados) y utilizaremos una imagen más complicada (con vello, bordes, etc.) para poder ver poco a poco cómo actúa cada uno de los filtros que componen el segundo procesado.

8.1-EJEMPLIFICACIÓN DEL PRIMER PROCESADO

Comenzaremos con una imagen sencilla, en la que encontraremos tan sólo piel, lunares y un poco de fondo:

Fig. 8.1.a: Imagen de ejemplo para el primer procesado.

Según se ha comentado anteriormente, los lunares cumplen una propiedad a tener en cuenta para diferenciarlos: son de color diferente a la zona de piel que los rodea, así que utilizaremos la detección de bordes como técnica para su detección. Seleccionamos el filtrado Prewitt que empíricamente resultó el más adecuado porque los demás mostraban detalles excesivos confundiendo marcas de la piel con la propia textura de ésta. Transformaremos primero la imagen en escala de grises y aplicaremos el filtro, dando como resultado la imagen siguiente:

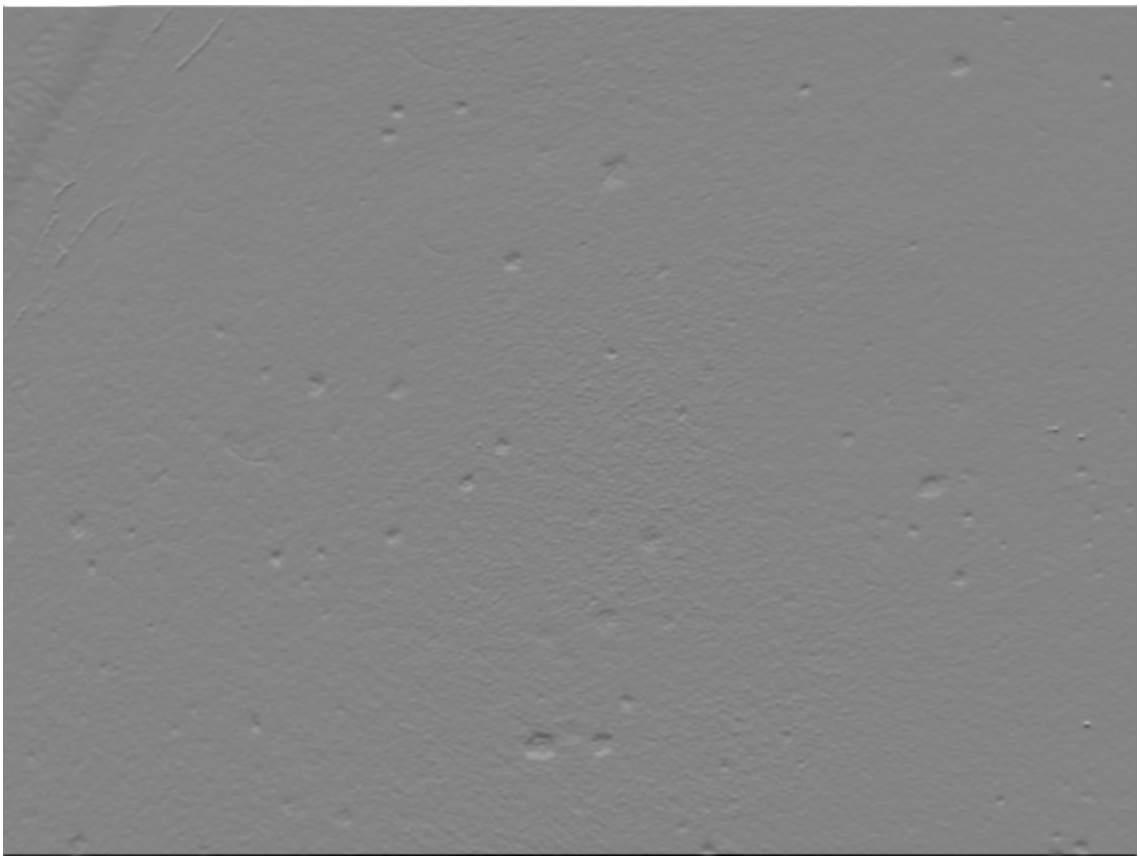


Fig. 8.1.b: Resultado para un filtrado Prewitt.

Resulta una imagen en escala de grises donde los valores más extremos corresponden a los bordes mientras que el resto de la imagen mantiene un nivel de gris constante. Deseamos utilizar un criterio para determinar si marcar o no marcar, es decir, necesitamos una imagen binaria (si un píxel es blanco lo marcaremos, si es negro no). Para ello debemos realizar una umbralización. Empíricamente se obtuvieron los mejores resultados acudiendo al histograma de la figura 8.1.a, calculando su máximo (correspondiente a la zona de las imágenes que no son bordes) y poniendo dos umbrales, uno sumando 10 a este máximo y otro restando 10 a este máximo. Los valores que se encuentren fuera de este rango (entre máximo-10 y máximo+10) serán marcados como blancos y los que se encuentren dentro como negros. A continuación se muestra un histograma típico de una imagen correspondiente a un filtrado Prewitt, nos hemos centrado en mostrar los niveles de gris que más se repiten en la imagen, y los valores comprendidos entre máximo-10 y máximo+10, consiguiendo separar la mayoría de los píxels que no pertenecen a los bordes:

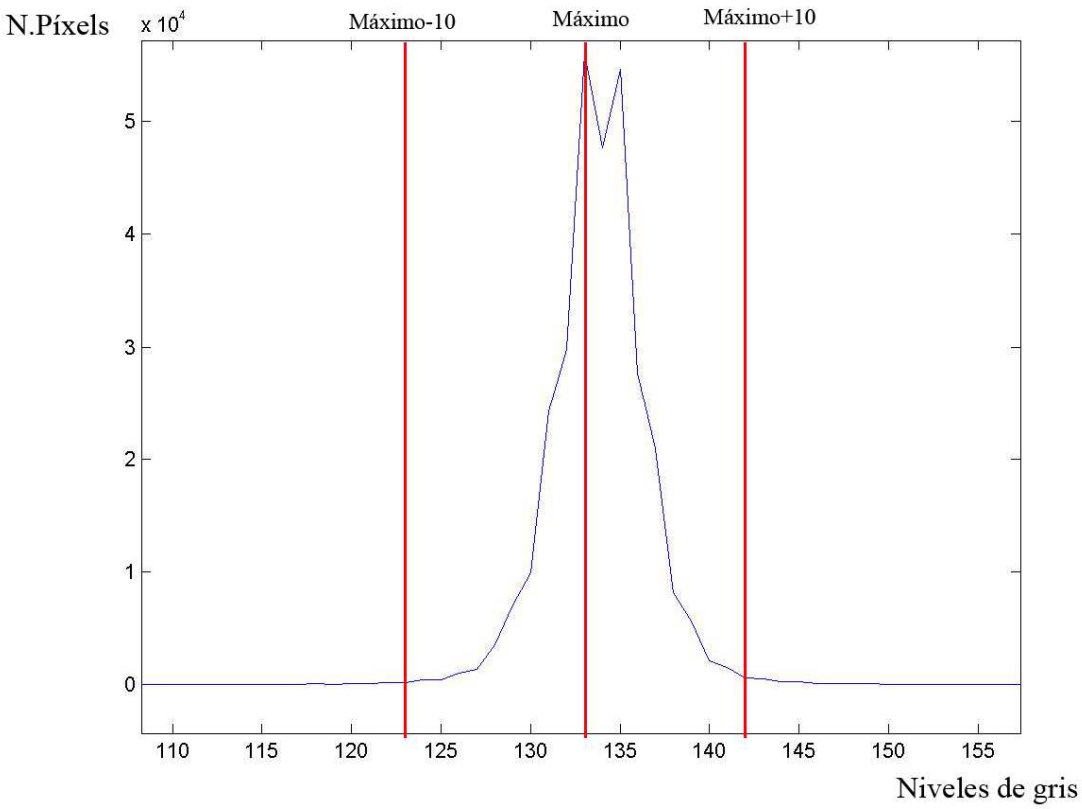


Fig. 8.1.c: Histograma típico de un filtrado Prewitt.

El resultado de aplicar esta umbralización será el de la figura 8.1.d:

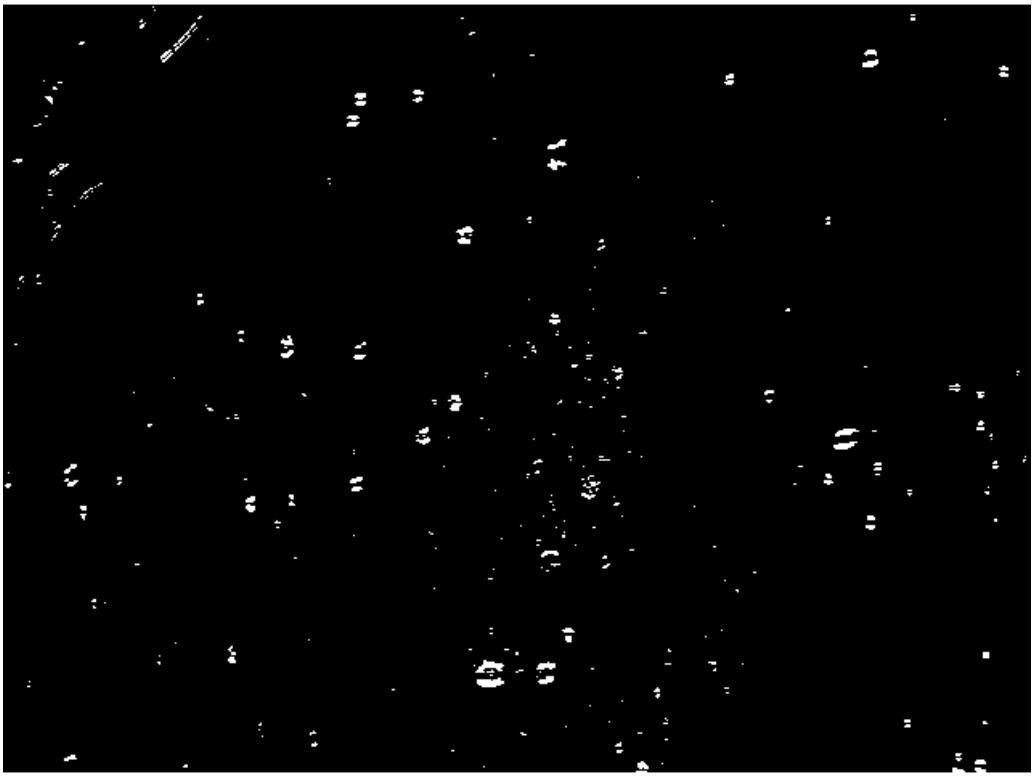


Fig. 8.1.d: Umbralización de un filtrado Prewitt.

Como se observa en esta imagen de cada marca se han obtenido los bordes superior e inferior. Esto se debe a que se ha utilizado el filtro Prewitt para bordes horizontales. Se repetirá el proceso para obtener las imágenes binarias de los bordes verticales, con 45º y con 135º obteniendo los resultados mostrados en las figuras 8.1.e, 8.1.f y 8.1.g:

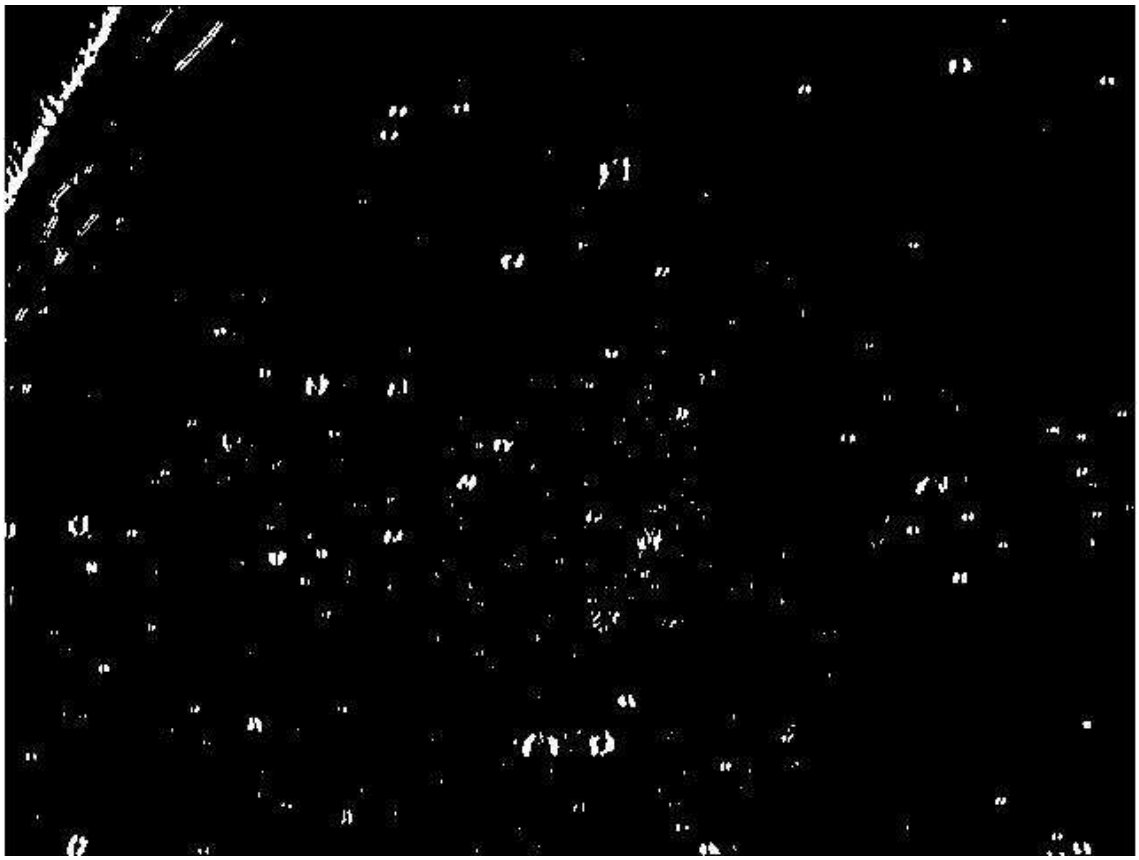


Fig. 8.1.e: Filtrado Prewitt de 45º umbralizado.

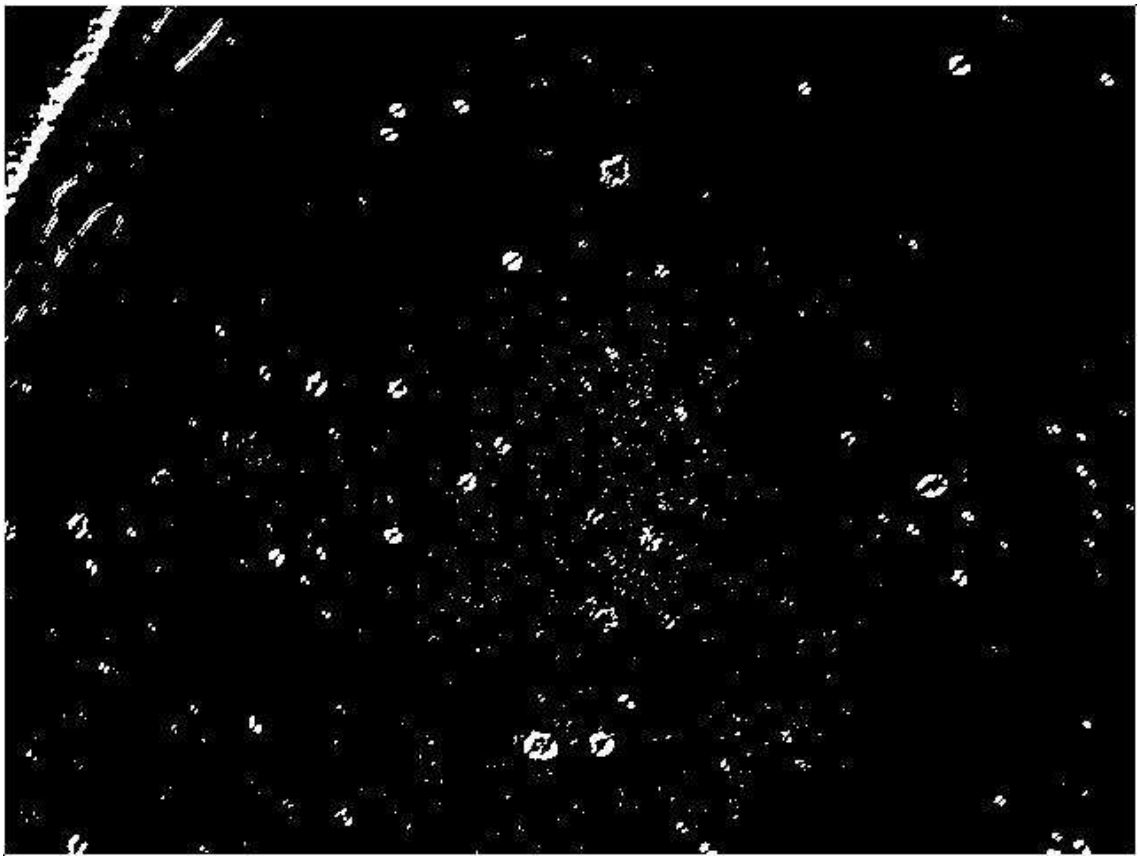


Fig. 8.1.f: Filtrado Prewitt de 45º umbralizado.

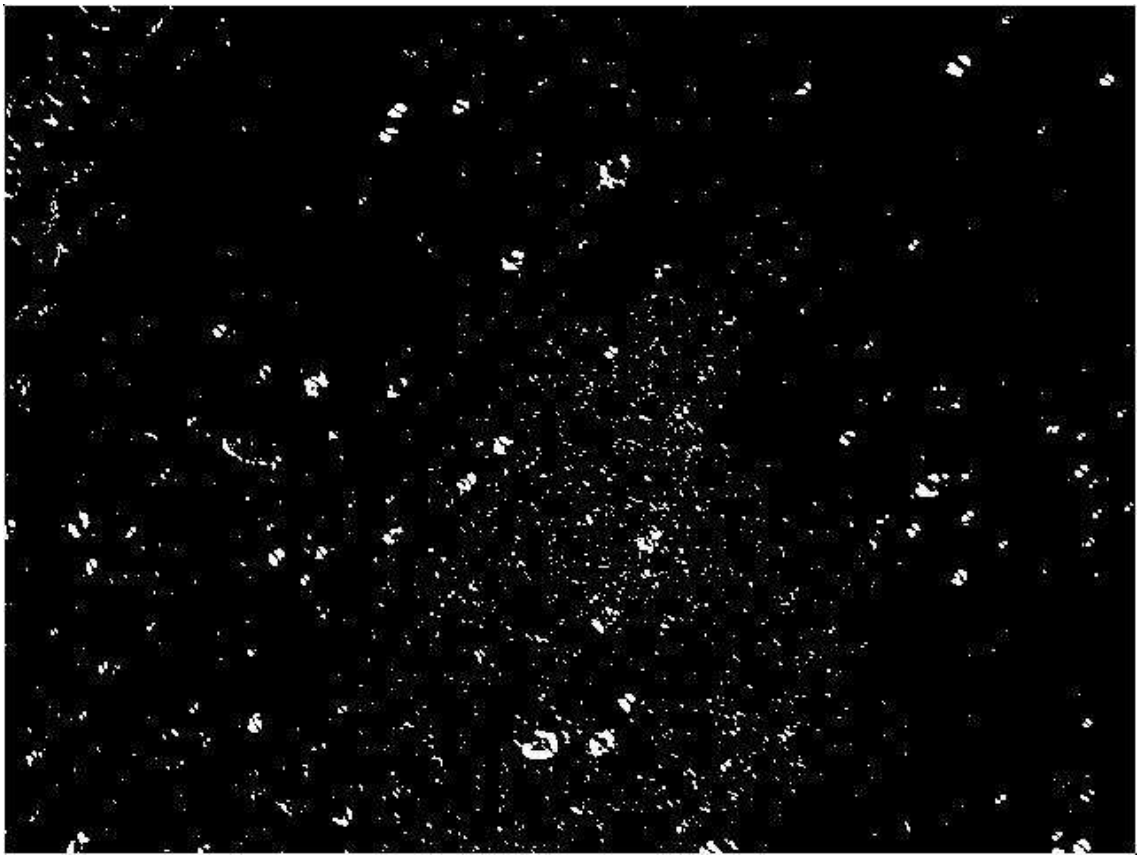


Fig. 8.1.g: Filtrado Prewitt de 45º umbralizado.

Para finalizar se harán barridos de las 4 imágenes binarias, marcando en la imagen original de color verde los píxels que en las imágenes binarias son de color blanco. El resultado será el mostrado en la figura 8.1.h:

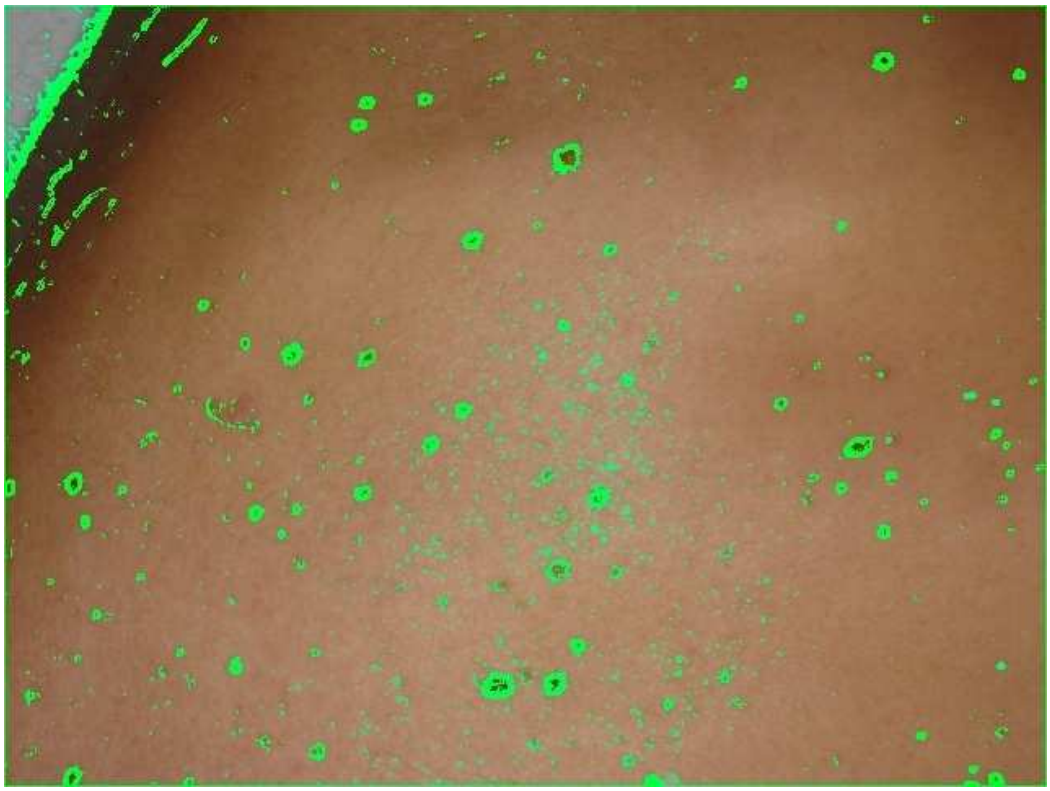


Fig.8.1.h: Resultado de aplicar el primer procesado a la imagen de ejemplo.

Los resultados para este primer procesado son bastante buenos pero hará falta un segundo procesado, sobre todo para las imágenes que tengan vello. En este primer paso tan sólo hemos marcado los bordes presentes en la imagen, sin tener en cuenta que muchos de estos bordes corresponden a vello, textura, contorno del cuerpo, etc.

8.2-EJEMPLIFICACIÓN DEL SEGUNDO PROCERSADO

Los resultados para el ejemplo anterior son bastante buenos pero ¿qué resultados obtendríamos en imágenes con vello, con bordes en brazos, textura de la piel marcada, etc.? Hemos seleccionado una imagen donde podemos encontrar problemas de este tipo:

Fig. 8.2.a: Ejemplo para el segundo procesado.

Tras aplicar el procesado visto anteriormente, el resultado obtenido será el de la figura 8.2.b:

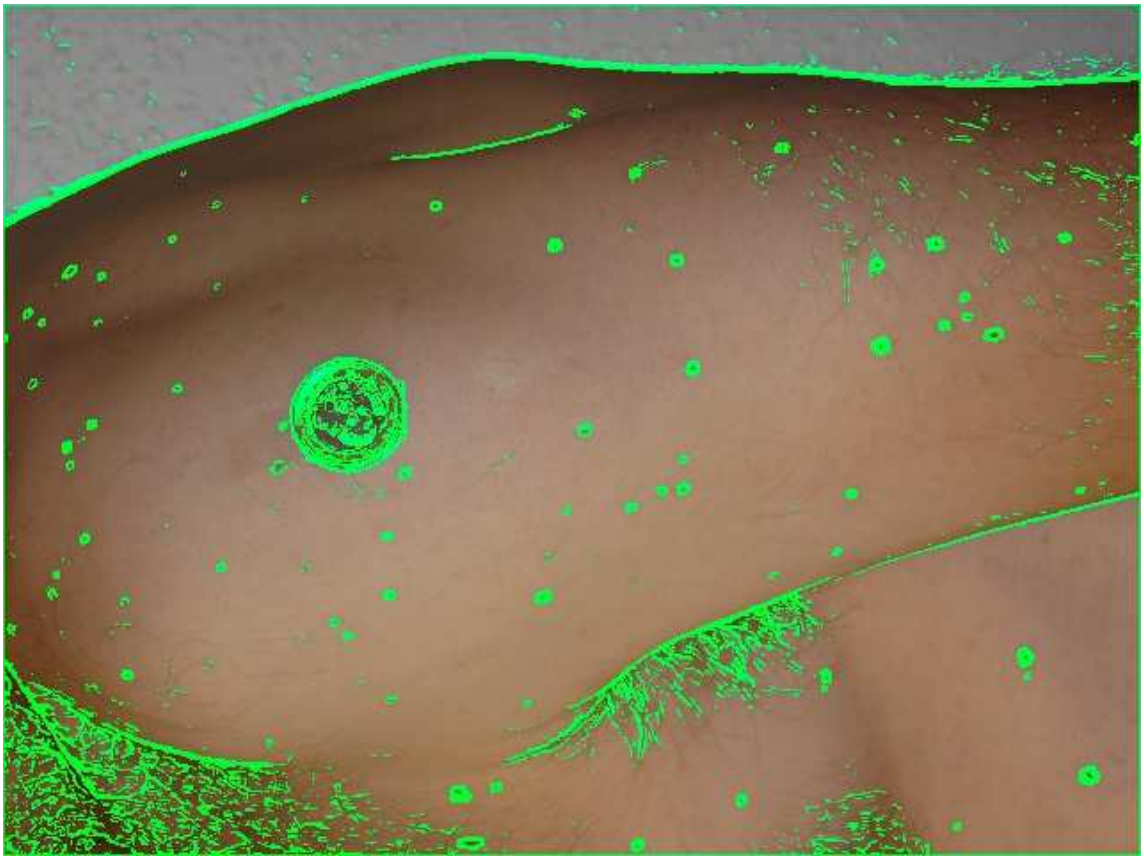


Fig. 8.2.b: Resultado tras el primer procesado.

Como se aprecia en la imagen, hay muchas marcas erróneas, desde marcas producidas por vello, bordes del brazo, límites de la imagen, etc. Por ello se decidió incluir un segundo procesado que eliminaría marcas mediante la aplicación diferentes filtros. El primero que se aplicará servirá para eliminar el marco de la imagen. El resultado se puede observar en la figura 8.2.c:

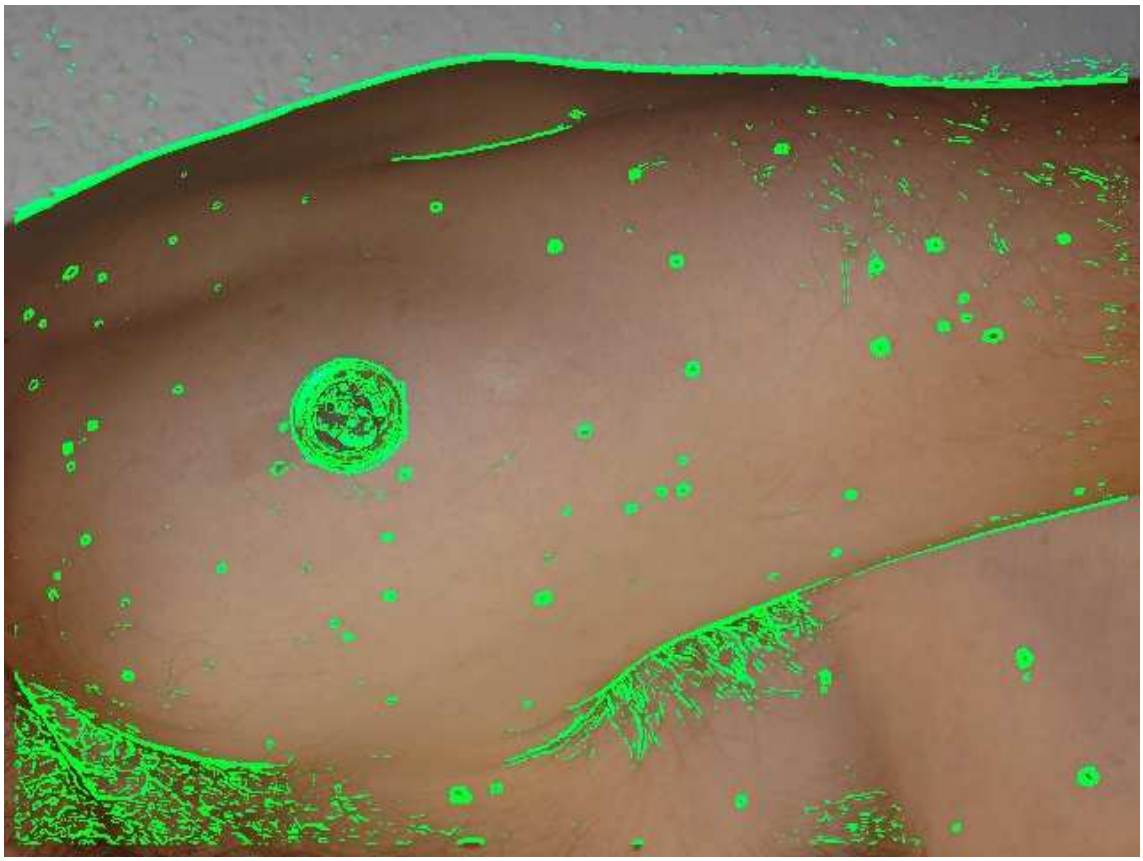


Fig. 8.2.c: Eliminación del marco de la imagen.

En la imagen se puede ver como un marco de 6 píxels de color ha sido eliminado de la imagen. El siguiente paso será aplicar un filtro que elimine marcas que no puedan ser lunares por sus características de color. Para ello se realizaron varios estudios de color que sirvieron para desechar algunas de las marcas. En primer lugar, se eligieron 20 puntos de 20 lunares al azar para comprobar sus niveles RGB, Gris y HSV. Se calcularon sus máximos y mínimos y se comprobó que seguían algunas pautas. En todos los casos se cumplía que: los niveles de rojo se encontraban dentro de los rangos 83-159, los de verde entre 38-106, los de azul entre 19-76, los de gris entre 54-107, los de S entre 33-88, los de V entre 28-82. Se calculó que el nivel de rojo era superior al 80% de la suma de los niveles de verde y azul. En cuanto a los niveles de H, eran superiores a 354 o inferiores a 2, es decir tienen matices rojos (el matiz va de 0 a 354, pero es cíclico, los valores muy altos o muy bajos corresponden a matices rojos). El resultado se puede observar en la figura 8.2.d:

Fig. 8.2.d: Resultado tras desechar marcas por sus características de color.

Este filtro ha conseguido un buen resultado para esta imagen. Se ha conseguido eliminar la mayoría de marcas pertenecientes a la moneda, pequeñas marcas insignificantes producidas por la textura de la piel y las marcas producidas por las fronteras entre la piel y el fondo. El siguiente filtro realizará una estimación de lo que pertenece a la piel y el fondo, para eliminar marcas producidas fuera de la zona de la piel. Para ello realizamos un estudio sobre el color de la piel aplicando el mismo sistema seguido con las marcas. Esto nos permitirá diferenciar la piel del fondo de la imagen para poder desechar los bordes producidos entre la piel y el fondo. Comprobamos que los colores también se encuentran en rangos comprendidos entre máximos y mínimos que podremos utilizar para diferenciar la piel. En el detector desarrollado, se restará al mínimo un valor de 10 y se sumará al máximo un valor de 10 para establecer un margen que solventará posibles errores en las medidas y que empíricamente han resultado ser mejores. Aplicado a la imagen de ejemplo, el resultado estimado será el mostrado en la figura 8.2.e:

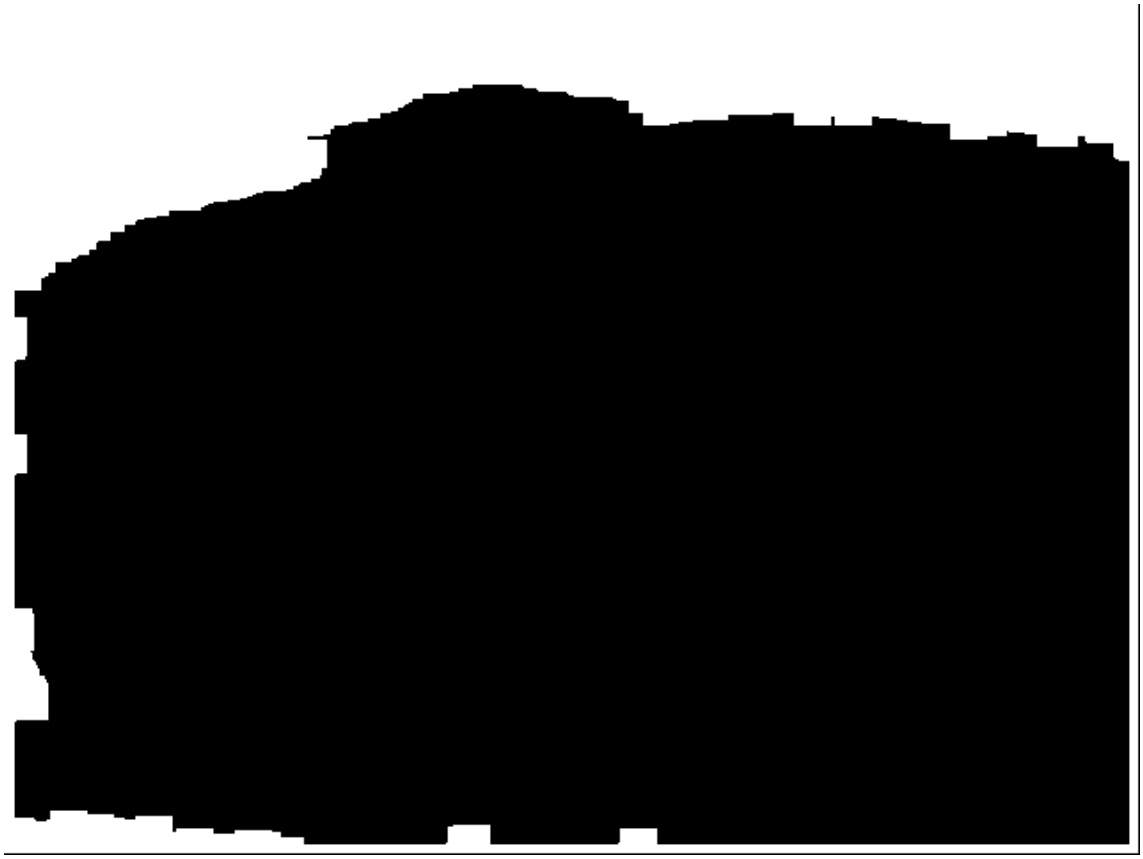


Fig. 8.2.e: Estimación del fondo para la imagen ejemplo.

La zona negra es nuestra estimación de la zona correspondiente a la piel, la zona blanca será el fondo. La estimación es buena porque incluye como fondo la zona grisácea que se encuentra en la zona superior de la imagen. También ha incluido parte de vello de la zona inferior. Tras eliminar las marcas correspondientes a la zona blanca, obtendremos el siguiente resultado:

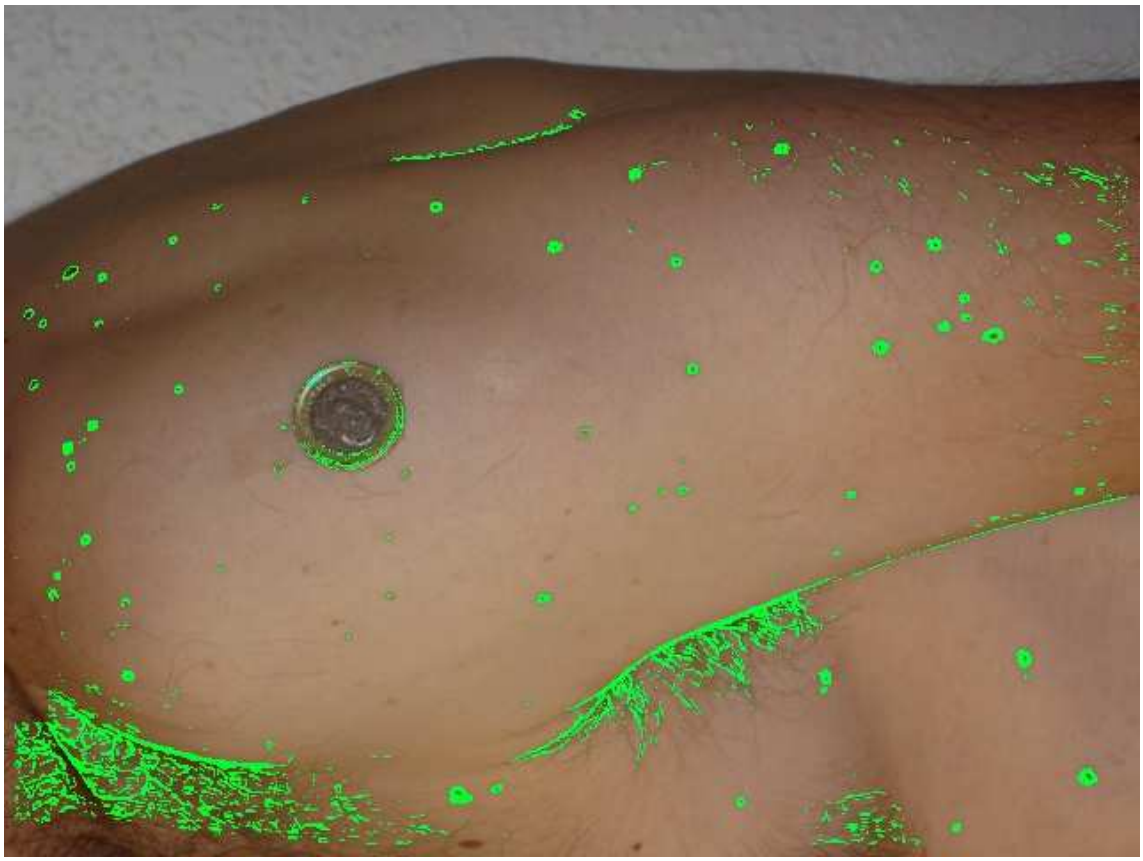


Fig. 8.2.f: Resultado tras desestimar el fondo de la imagen.

La imagen no ha cambiado mucho, se han eliminado zonas de vello y alguna pequeña marca situada en la frontera entre la espalda y el fondo. Se debe a que la mayoría de las marcas de la frontera y del fondo habían sido eliminadas ya por el filtro anterior. A continuación aplicaremos el filtro para el vello (lo llamaremos filtro P). Para comprender el funcionamiento del filtro P, tendremos que estudiar el algoritmo (ver el apartado). La estimación para el vello obtenida para esta imagen será la mostrada en la figura 8.2.g:

Fig. 8.2.g: Estimación de vello para la imagen de ejemplo.

El resultado no es óptimo porque en esta imagen el vello se presenta muy fino y no como grandes masas como aparecen otras imágenes. De todas formas el filtro ha conseguido encontrar parte del vello de la zona inferior izquierda y parte del vello de la axila. Para esta imagen en concreto, obtendremos mejores resultados con el último filtro implementado, uno basado en el tamaño y la forma de los lunares (lo llamaremos filtro B). Primero realizaremos una extracción de contornos y procederemos a su estudio. Para ello se obtuvieron los tamaños de 30 lunares al azar pero con tamaños significativos. Se calculó el tamaño en píxels tanto horizontal como verticalmente asignando como “lado largo” la mayor des estas distancias y como “lado corto” la menor de las distancias. Observamos que los lunares tendrán como mucho una longitud de 34 píxels (después se les dará un margen). Como poco tendrán una longitud de 4 y la diferencia entre el lado más largo y el más corto del lunar será como mucho de 10 píxels. El resultado obtenido para nuestro ejemplo tras la extracción y el estudio de los contornos se puede ver en la figura 8.2.h:

Fig. 8.2.h: Resultado tras la extracción y el estudio de contornos.

En este caso el filtro B ha funcionado muy bien. Nos ha permitido eliminar la mayor parte de las marcas que había producido el vello y que el filtro P no había conseguido eliminar por no encontrar el pelo como una gran masa. Además ha conseguido eliminar marcas de la axila y de la frontera entre el brazo y la espalda.

Para finalizar aplicaremos la extracción de contornos en la imagen sin aplicar los filtros P y B; y en la imagen con el filtro P para obtener los tres resultados. Se ha decidido hacerlo de esta manera, porque no se llegó a un compromiso que permitiera una óptima señalización de los lunares eliminando los errores producidos por vello y por otros bordes.

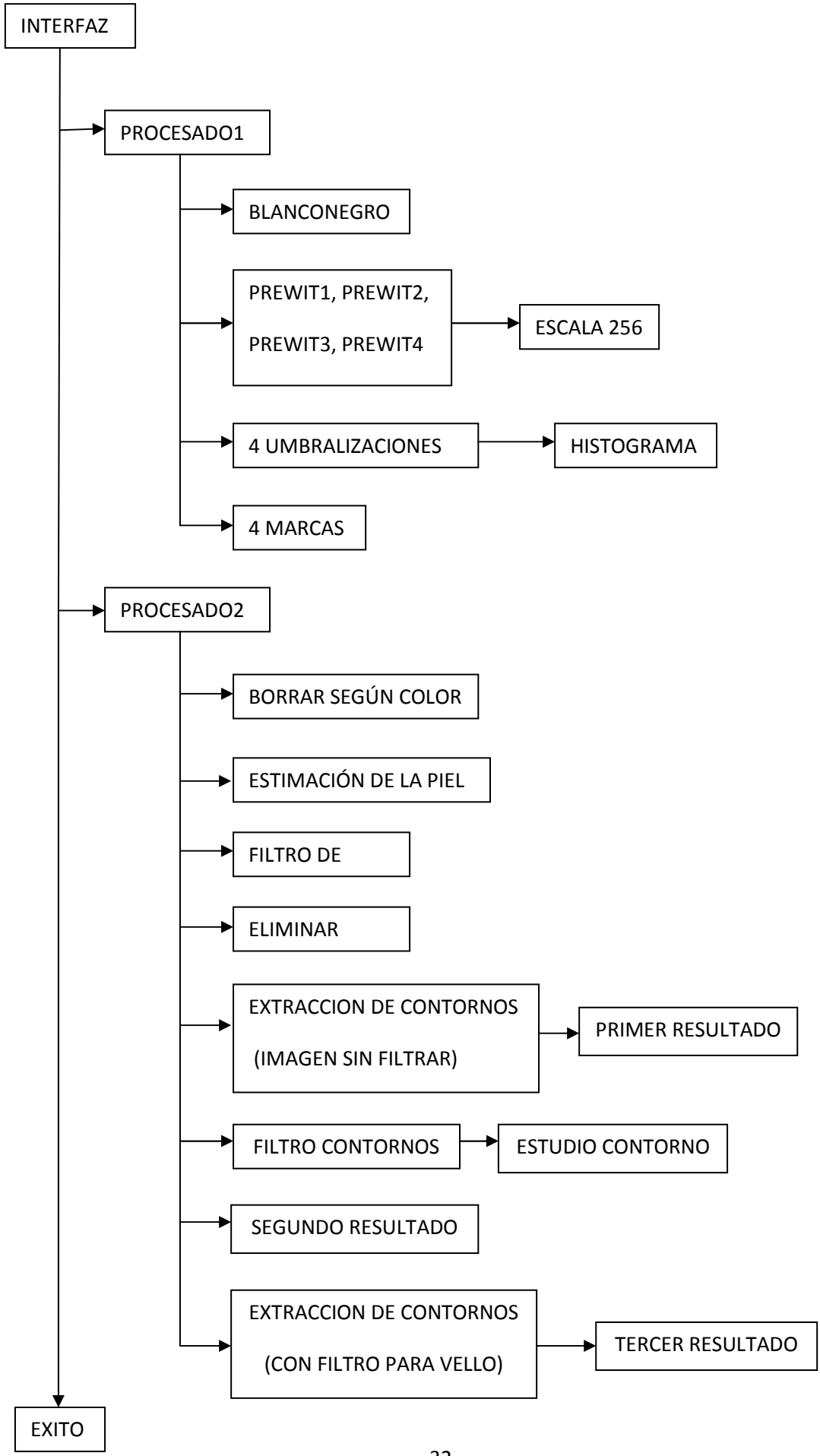
Aquí ha terminado todo el procesado, a continuación presentamos las tres imágenes resultado de aplicar los diferentes filtros:

Fig. 8.2.i: Resultado sin utilizar filtros.

Fig. 8.2.j: Resultado aplicando el filtro P.

Fig. 8.2.i: Resultado utilizando el filtro B.

Para finalizar este apartado, mostraremos un esquema general de los algoritmos que van ejecutándose a lo largo del programa. En los apartados posteriores explicaremos detalladamente y uno a uno cada uno de estos algoritmos. Puede observarse claramente como de una interfaz general, se hará la llamada a los dos procesados una vez introducida la imagen. Después cada uno de estos procesados irá llamando a las funciones necesarias para ir obteniendo los resultados esperados. En el primer procesado se realizará el filtrado Prewitt, la umbralización y el marcado de la imagen. Y en el procesado 2 se irá aplicando los diferentes filtros explicados anteriormente (vello, contornos,…) para finalmente lanzar la interfaz llamada “éxito” mostrando los diferentes resultados.



9-IMPLEMENTACIÓN DE LOS ALGORITMOS

9.1-INTERFAZ GRÁFICA Y PROGRAMA GENERAL

Para un uso más agradable e intuitivo del programa diseñado, se pensó en incluir una sencilla interfaz gráfica. El objetivo era que en ella se pudiera elegir la imagen a explorar y al terminar el procesado, se vieran los diferentes resultados. También se decidió que la interfaz mostrará la imagen procesada mientras se ejecuta todo el procesado y que se mostrará automáticamente un mensaje de error, en el caso de que el programa fallara o el usuario introdujera incorrectamente la imagen. El resultado para la pantalla inicial fue el siguiente:

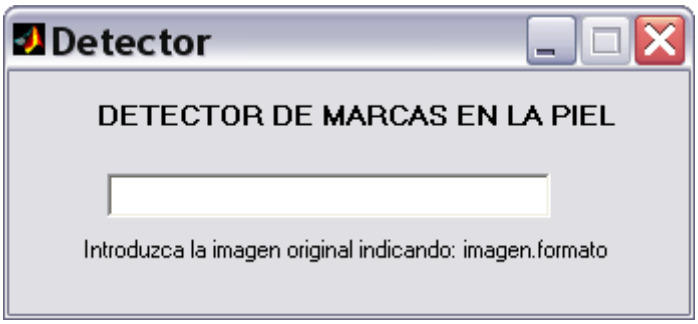


Fig. 9.1.a: Pantalla inicial.

Con la herramienta Guide se creó la interfaz que consta de la propia ventana (con nombre “Detector”), un texto estático con el título “Detector de marcas en la piel”, otro texto estático mostrando la forma de meter el nombre de las imágenes y una caja de texto editable para poner el nombre de la imagen y que tras pulsar Intro pondrá en funcionamiento el programa. Es necesario que la imagen se encuentre en el mismo directorio que el programa, además el directorio de trabajo en Matlab debe ser el mismo. En caso contrario, o en el caso de que la imagen se introduzca incorrectamente, el programa lanzará el aviso de error como veremos más adelante. Esta interfaz principal requería una programación representada en el siguiente esquema de la figura 9.1.b:

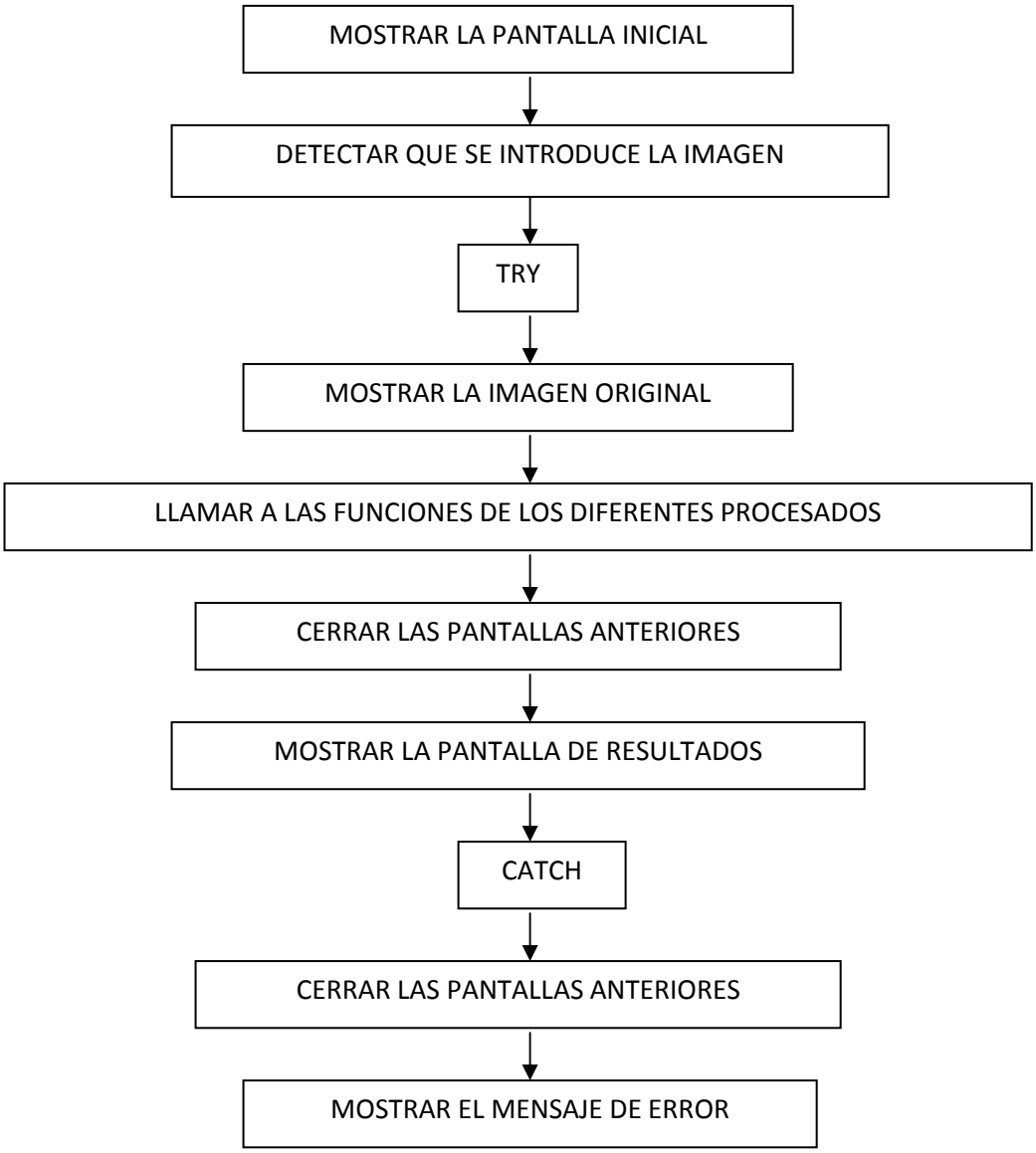


Fig. 9.1.b: Esquema del programa de la interfaz inicial.

En nuestra interfaz será necesario introducir el nombre de la imagen seguido de un punto y la extensión del formato de la imagen (bmp y jpg son los más comunes). Será necesario que la imagen se encuentre en el mismo directorio que el programa y que Matlab tenga este directorio seleccionado como directorio de trabajo. En el esquema se muestran dos comandos que son 'try' y 'catch', cuando el programa funcione correctamente se ejecutarán las instrucciones comprendidas entre ambos comandos. Primero se mostrará la imagen original mientras se llama a las diferentes funciones que componen todo el procesado. Después se cerrarán todas las ventanas anteriores y se mostrará la pantalla con todos los posibles resultados. En el caso de que alguna de estas instrucciones falle o que la imagen introducida sea errónea, se ejecutarán las instrucciones que siguen al comando 'catch'.

9.1.1-FUNCIONES QUE COMPONEN EL PROCESADO

Necesitaremos un programa principal que vaya ejecutando una a una todas las instrucciones necesarias para el correcto funcionamiento de todo el procesado. Como ya hemos visto, dividiremos todo el procesado en dos partes, una que haga una señalización inicial y otra que vaya desechando marcas según una serie de criterios. Nuestra función inicial también será capaz de esperar a que introduzcan la imagen, cargar la imagen introducida y dar los resultados una vez terminado el proceso. El esquema resultante será el siguiente:

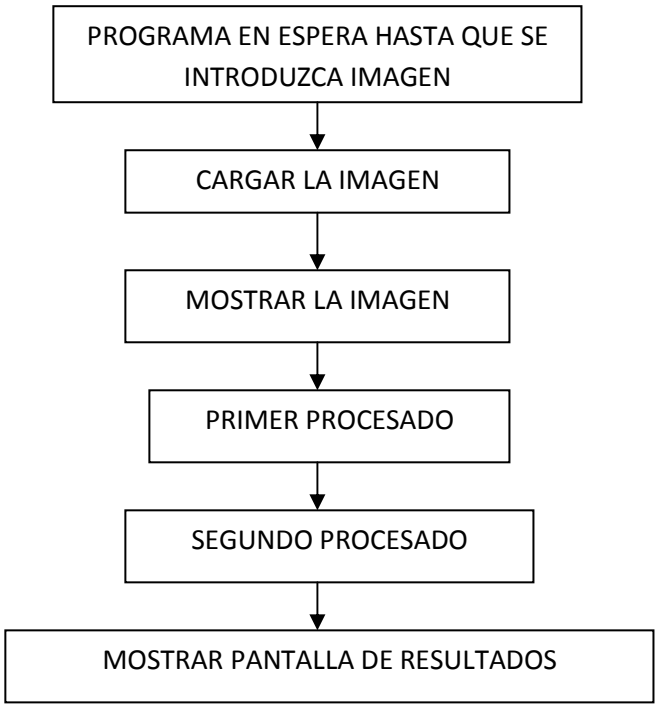


Fig. 9.1.1.a: Esquema del programa principal.

El programa principal esperará a que en la pantalla principal se introduzca el nombre de la imagen y se pulse la tecla intro. Entonces cargará la imagen y hará una copia para su procesamiento posterior. Mostrará la imagen inicial mientras se va realizando todo el procesado para que el usuario compruebe que es la imagen deseada la que ha introducido en el cuadro de diálogo. Se procederá entonces a realizar los dos procesados, para terminar mostrando los resultados cuando éstos terminen.

9.1.2-PROGRAMA ÉXITO

Este programa también es parte de la interfaz gráfica y se ejecutará al final del procesado (si no ha habido ningún error) y se cerrarán el resto de ventanas. El programa sirve para informar al usuario, acústica y visualmente, de que el procesado ha concluido con éxito, además de permitirle elegir entre cinco opciones (dispuestas en cinco botones diferentes): permite ver la imagen original, el resultado sin utilizar los filtros, el resultado utilizando el filtro P o B y finalmente permite abrir otra vez el programa interfaz para procesar una nueva imagen. La ventana presentará la siguiente apariencia:

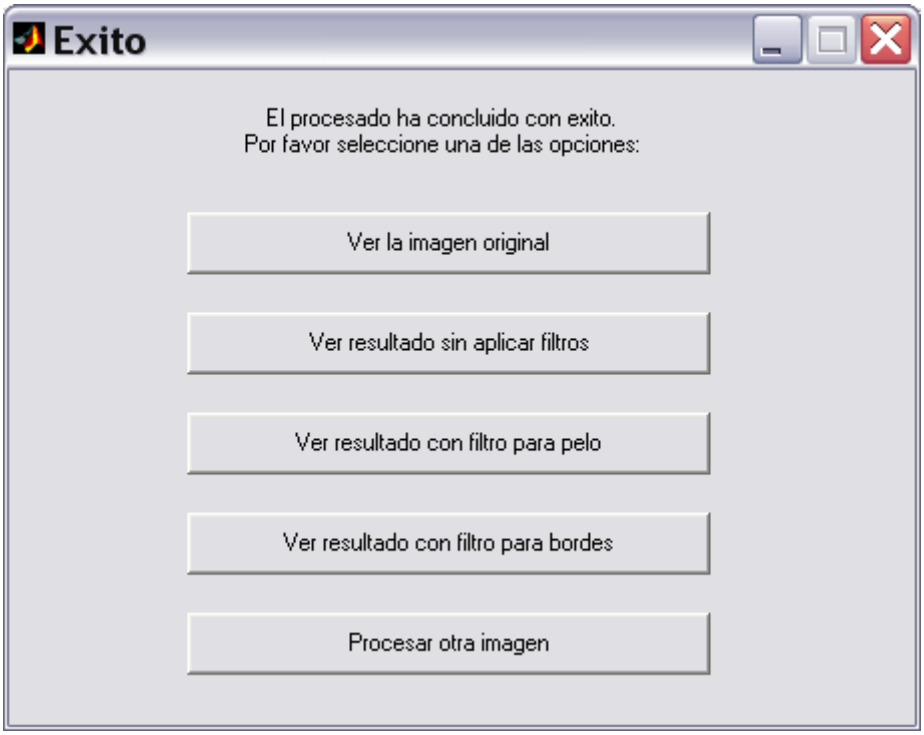


Fig. 9.1.2.a: Ventana del programa “éxito”.

El algoritmo necesario, debe ser capaz de detectar qué botón ha pulsado el usuario para actuar en consecuencia, ya sea mostrando la imagen correspondiente o volviendo a la pantalla de introducción de la imagen a procesar para analizar una imagen diferente. El esquema será el mostrado en la figura 9.1.2.b:

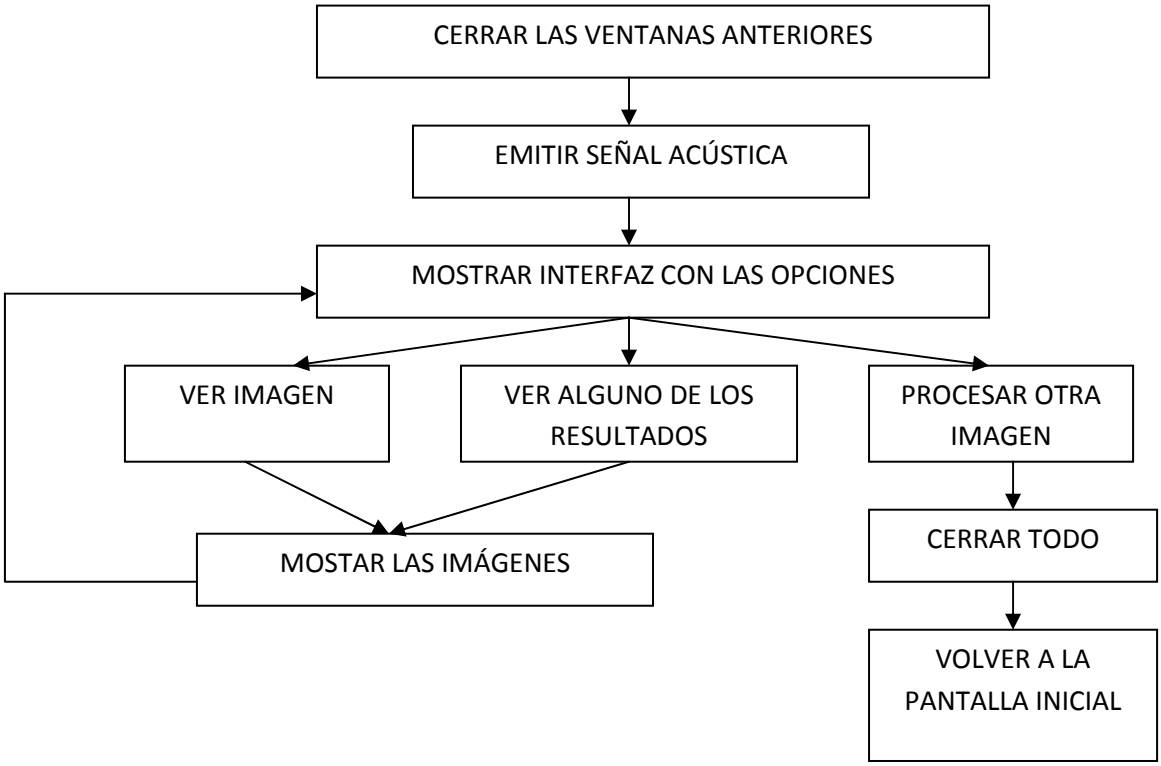


Fig. 9.1.2.b: Esquema del programa “éxito”.

El programa debe permitir ver simultáneamente las imágenes originales y procesadas para comprobar resultados, hacer comparaciones, etc. También permitirá salir de la aplicación en cualquier momento al pulsar el aspa (como la mayoría de las aplicaciones para Windows).

9.1.3- PROGRAMA ERROR_EN_IMAGEN

Tras producirse un error el programa principal saltará a las instrucciones comprendidas entre Catch y End, entonces será cuando se ejecute este programa. Emitirá un sonido e indicará al usuario en una ventana, que se ha producido un error y que se cerciore de que ha introducido correctamente la imagen y que la imagen se encuentra en el directorio de trabajo. El mensaje de error será el siguiente:

Fig. 9.1.3.a: Mensaje de error.

Tras ejecutarse se cerrarán todas las ventanas. Este es el último elemento de la interfaz gráfica. A continuación pasaremos a mostrar el funcionamiento del algoritmo para el primer procesado.

9.2-PRIMER PROCESADO

Como se explicó en el punto 2 dedicado al diseño del procesado de imagen tras obtener información de éstas, primero realizaremos un procesado (implementado en el programa procesado1) que realice el filtrado Prewitt en las cuatro direcciones, que aplique las umbralizaciones y que haga una primera señalización en la imagen original guardando los resultados. El esquema del programa será el siguiente

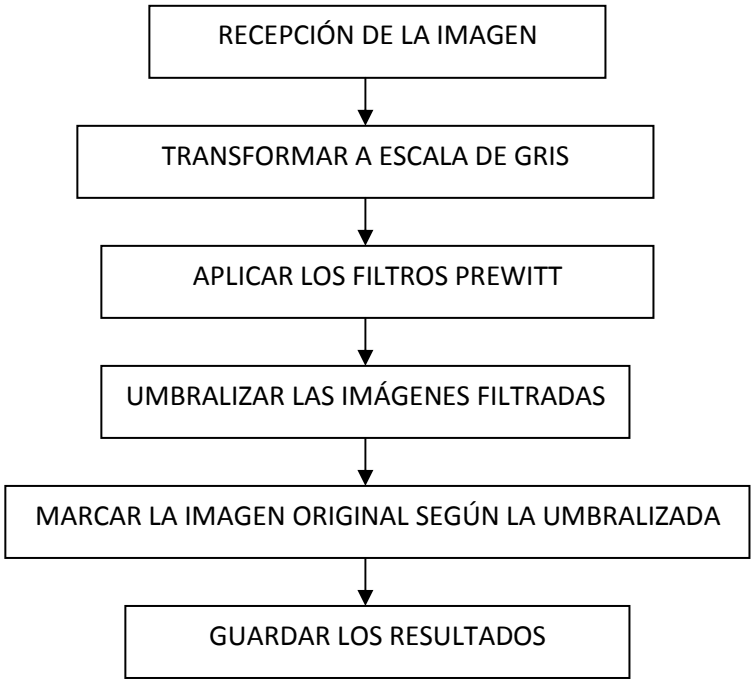


Fig. 9.2.a: Esquema del primer procesado.

El programa necesita la matriz de la imagen original, que se la proporciona el programa general interfaz. El siguiente paso es transformarla en una imagen en escala de grises a la que se le puedan aplicar los filtros Prewitt para la detección de bordes. A esta imagen se le aplicarán cuatro máscaras correspondientes a las cuatro direcciones Prewitt distintas: Horizontal, vertical, 45º y 135º.

Después de esto se aplicará una umbralización a cada uno de los resultados obtenidos y para terminar se irá marcando la imagen original progresivamente según estos umbrales y se guardara el resultado en el directorio de trabajo como “Procesado1.bmp”. A continuación se irán explicando cada parte del procesado con más precisión.

9.2.1-TRANSFORMACIÓN DE RGB A ESCALA DE GRISES

La función blanconegro permite transformar una imagen en color a su versión en escala de grises, que nos permitirá aplicarle el filtrado prewitt. Para ello es necesario darle la matriz de la imagen original en formato RGB.

A continuación se recorrerá cada píxel de la imagen, extrayendo la componente de R, G y B para cada píxel. Se realizará una media entre los tres valores para el cálculo del nivel de gris. Después se realiza un redondeo (las imágenes no pueden tener componentes decimales, sólo enteros comprendidos entre 0 y 255).

9.2.2-FUNCIONES PREWITT

El siguiente paso será aplicar los filtros para la detección de bodes. Como se explicó en el apartado 2, aplicaremos 4 filtros Prewitt: Horizontal, vertical, 45º y 135º, cuyas máscaras son:

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

-1	-1	0
-1	0	1
0	1	1

0	-1	-1
1	0	-1
1	1	0

Fig. 9.2.2.a: Máscaras correspondientes a los filtros Prewitt.

Han sido implementadas en 4 funciones respectivamente: Prewitt1, Prewitt2, Prewitt3 y Prewitt4. Se decidió no utilizar la función "filter2" de Matlab, para poder realizar directamente un redondeo y un cambio de escala a los niveles de gris. El esquema de funcionamiento será el mismo para las 4 funciones y sólo cambiarán los coeficientes a tener en cuenta de los píxels vecinos para el cálculo del filtro. En la figura 9.2.2.b se muestra su modo de funcionamiento:

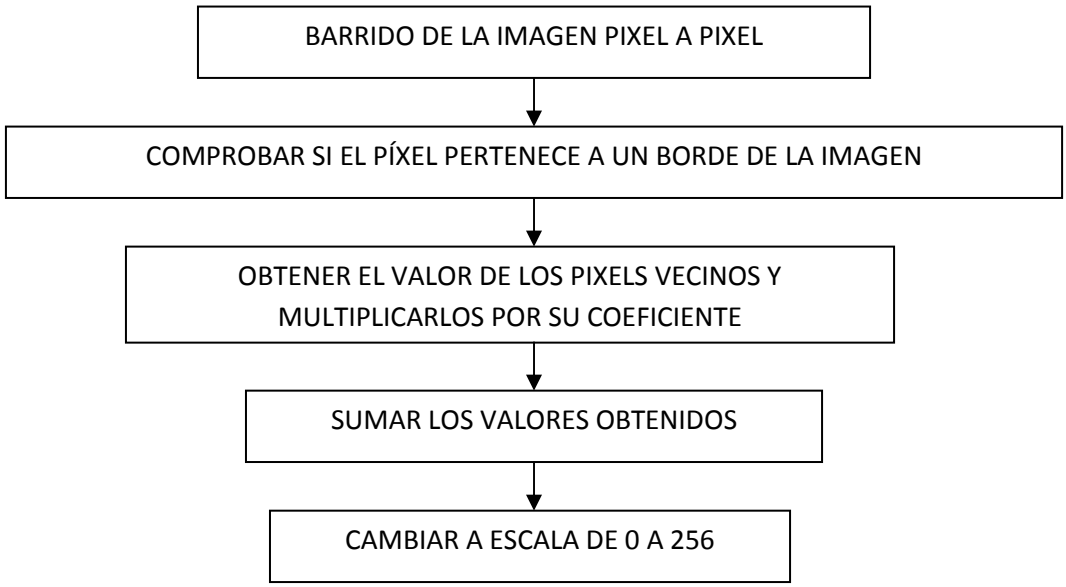


Fig. 9.2.2.b : Esquema para las funciones Prewitt.

Se realizará un recorrido píxel a píxel de la imagen y antes de nada se comprobará si el píxel cuyo valor se quiere calcular se sitúa en alguno de los bordes de la imagen. En el caso de que corresponda a uno de los bordes, no se tendrán en cuenta todos los vecinos, ya que al situarse en un borde, no tendrá vecinos en alguna de las direcciones y la función nos mostraría un error de fuera de rango. Para ejemplificarlo: supongamos que un píxel se encuentra en la fila número 1 (uno de los bordes), se necesitarán píxels de la fila número 0 que no existe (las filas van desde 1 hasta M), por tanto el programa dará un error que indica que se intenta acceder a unas coordenadas inexistentes en la matriz. Para evitarlo en los píxels de los bordes sólo utilizaremos para el cálculo 5 de los 8 vecinos, en nuestro ejemplo los píxels que correspondieran a la fila 0 no serían tenidos en cuenta, mientras que el resto, sí. Una vez hecha la comprobación, se obtendrán los valores para los píxels vecinos a tener en cuenta y se multiplicarán por el coeficiente correspondiente, indicado por la tabla del filtro Prewitt a aplicar. Sumaremos todos los valores para obtener el valor del filtro Prewitt para ese píxel en cuestión.

A modo de ejemplo, supongamos que queremos aplicar el filtro Prewitt horizontal para un píxel, la máscara correspondiente será la siguiente:

-1	-1	-1
0	0	0
1	1	1

Fig. 9.2.2.c: Máscara del filtro Prewitt horizontal

Para el cálculo del valor del filtro para el píxel central, sumaríamos los valores de los vecinos de la fila siguiente y les restaríamos los vecinos de la fila anterior. Tras realizar los cálculos nos encontraremos con un problema: los valores obtenidos para el filtro superan el rango dinámico de grises de 0 a 255. Se debe asignar a cada nivel de gris un valor entero comprendido entre 0 y 255 (una escala de 256 valores, 1 Byte) Y para eso se ha creado la función Escala 256. Su funcionamiento es sencillo. Primero se calculan los valores máximo y mínimo obtenido en el filtro. Después se aplicará para cada píxel del filtro la siguiente fórmula:

$$\text{Valor del píxel} = \frac{(ValorActual - Mínimo) * 255}{Máximo - Mínimo}$$

Fig. 9.2.2.d: Fórmula para el cambio de escala.

Para finalizar se aplica un redondeo, puesto que los valores para una imagen no pueden contener números decimales. Se puede observar a continuación un ejemplo de la imagen resultante:

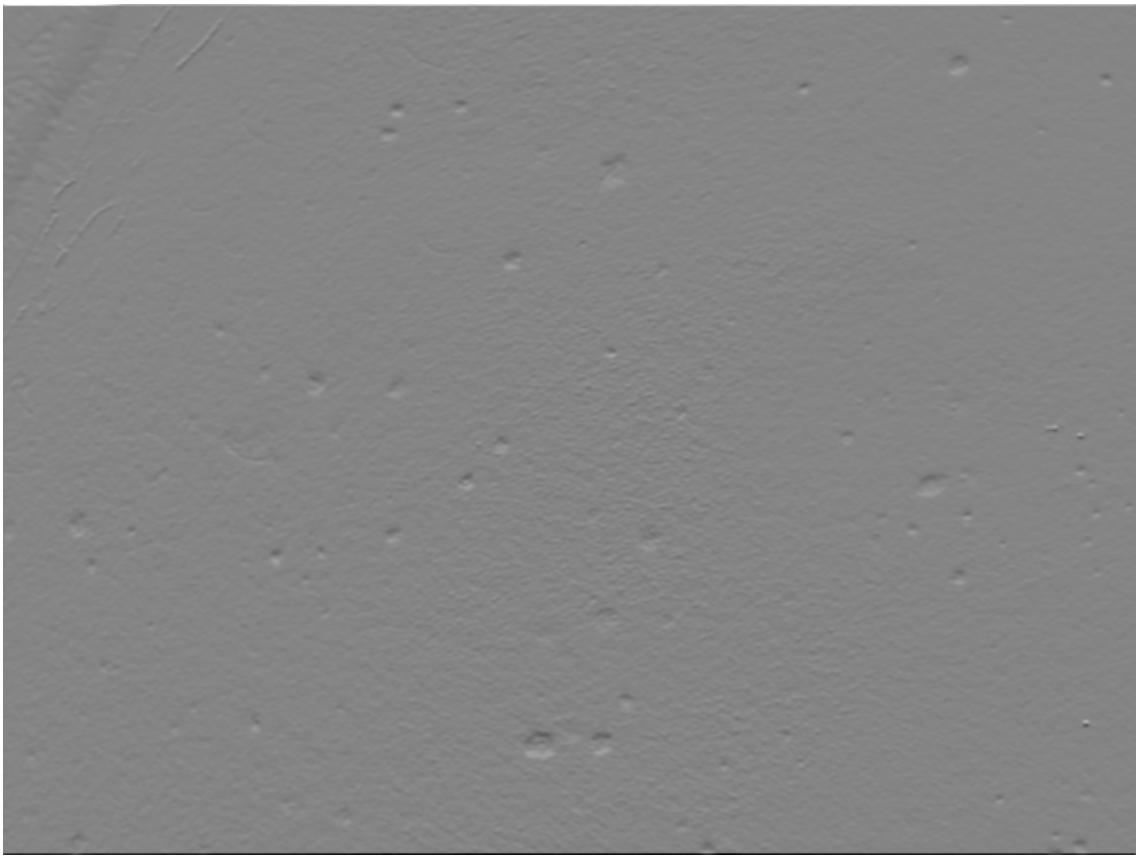


Fig. 9.2.2.e: Ejemplo de filtro Prewitt.

Tras obtener los resultados para los 4 filtros, se deberán umbralizar y marcar por separado, obteniendo el resultado para el primer procesado.

9.2.3-FUNCIÓN UMBRALIZAR

Una vez obtenido el filtro Prewitt, tendremos que decidir que píxels deben ser marcados y cuáles no. Para ello, realizaremos una umbralización sobre cada uno de los 4 filtros obtenidos anteriormente. Obtendremos una imagen binaria, que utilizaremos para realizar una marca sobre la imagen original, señalando los lunares (cuando un píxel de la imagen binaria sea blanco, marcaremos el píxel correspondiente en la imagen original). El esquema de funcionamiento de la umbralización será el siguiente:

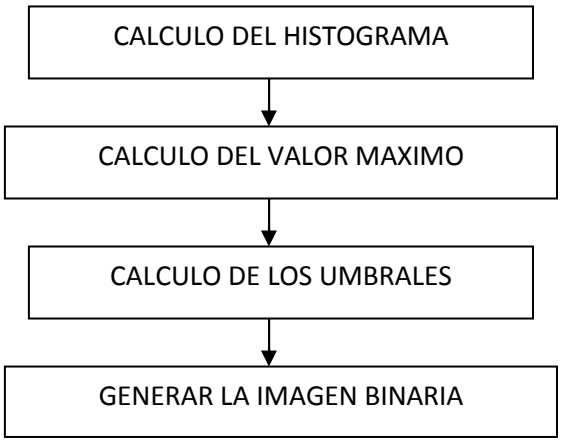


Fig. 9.2.3.a: Esquema de la función “umbralizar”.

En primer lugar la función obtiene el histograma de la imagen y su máximo (Después se explicará el funcionamiento de estas funciones). Luego sobre este máximo se generan dos umbrales el máximo-10 y el máximo+10 puesto que los valores correspondientes a los bordes se escaparán de este rango. Observando la imagen mostrada en el apartado anterior se comprueba que la mayoría de píxels se encuentra en un rango perteneciente a niveles medios de gris (comprendidos entre los valores dados anteriormente), mientras que los bordes serán la minoría de píxels, como se muestra en este ejemplo de histograma en la figura 9.2.3.b:

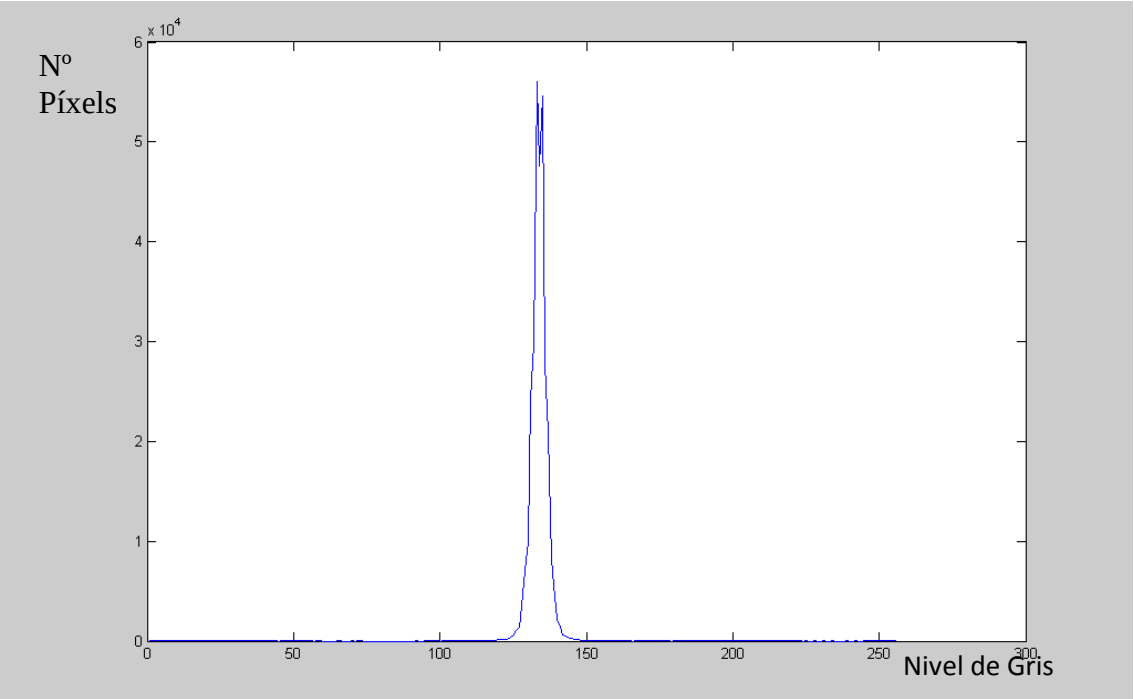


Fig. 9.2.3.b: Histograma de un filtrado Prewitt.

El pico representa los niveles de gris que más abundan en la imagen, que como se observa comprende un rango desde el nivel de gris con más píxels-10, hasta el nivel de gris con más píxels+10. El resto de niveles de gris pertenecen a los bordes.

Para generar la imagen binaria, recorreremos cada píxel del filtro y si el nivel de gris esta fuera del rango será marcado como blanco y si esta dentro como negro, generando así una imagen binaria preparada para marcar sobre la imagen original. Obtendremos un resultado similar al de la figura 9.2.3.c:

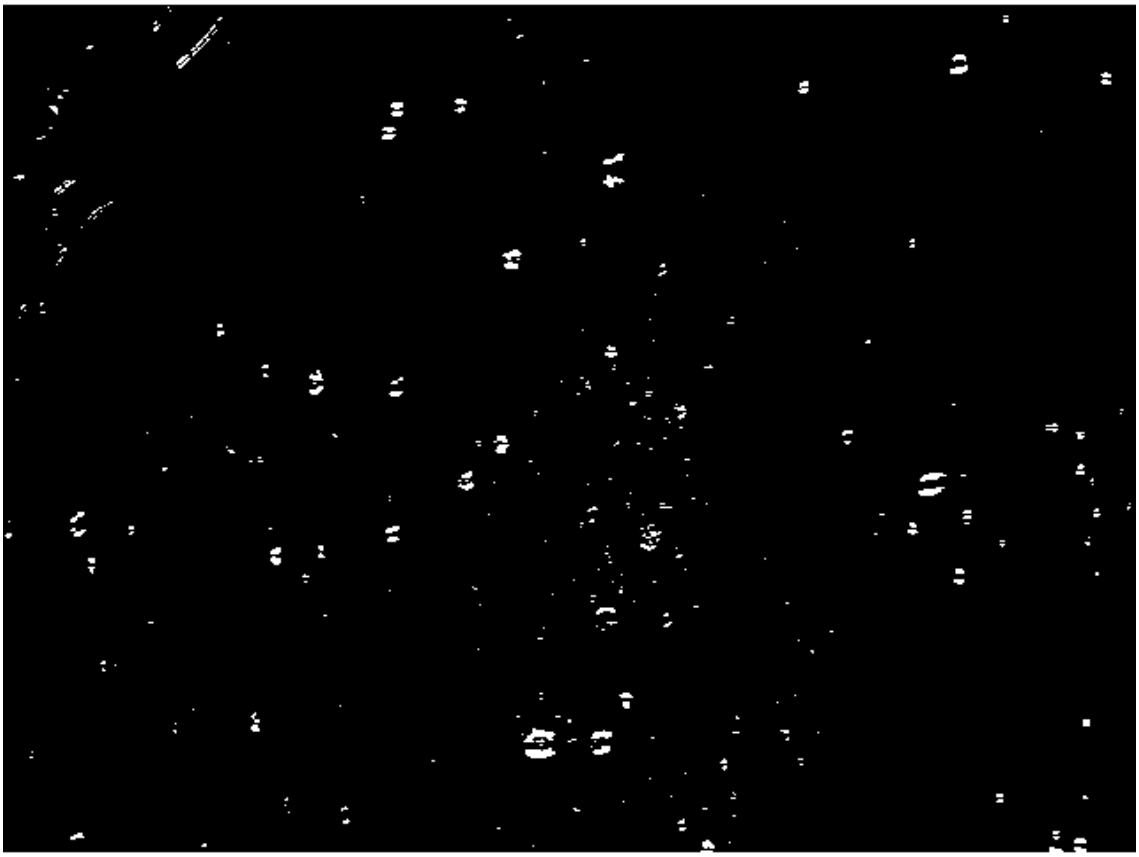


Fig. 9.2.3.c: Ejemplo de imagen binaria lista para marcar

Antes de continuar con el procesado, se debe explicar el funcionamiento de la función histograma. Se implementó está función porque la versión de Matlab disponible en el momento del desarrollo, no incluía la función “Imhist” que hubiera permitido realizar este mismo trabajo. El esquema de la función es el mostrado en la figura 9.2.3.d:

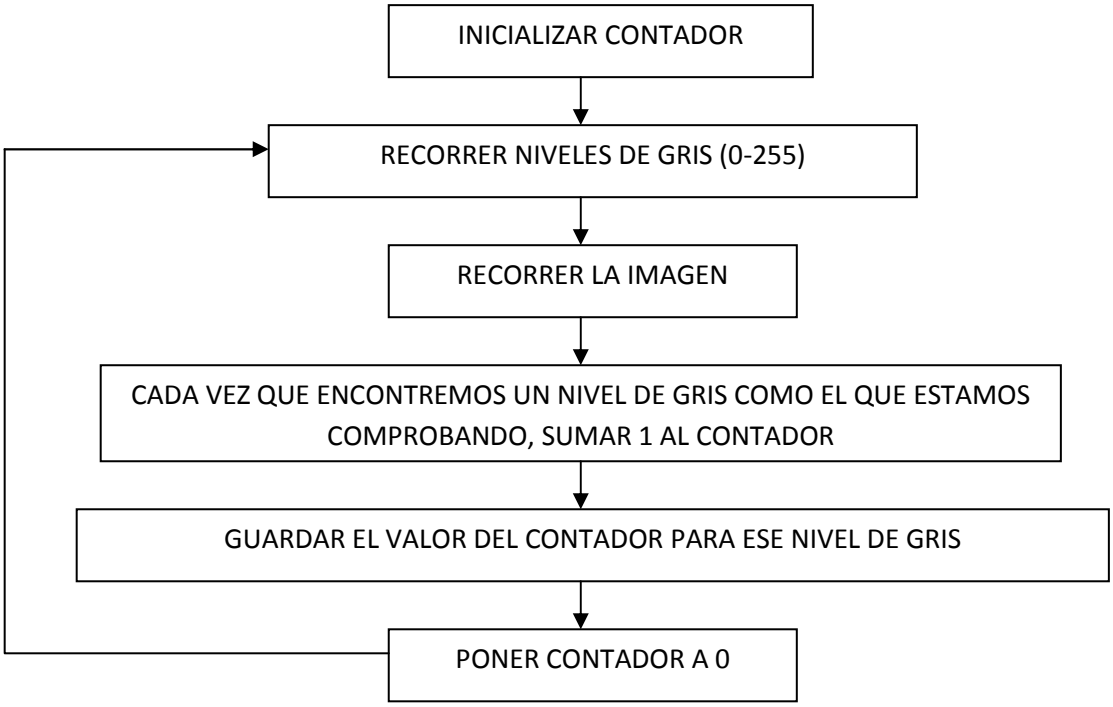


Fig. 9.2.3.d: Esquema de la función “histograma”.

En primer lugar se inicializará el contador a 0. Después se barrerá toda la escala de 256 valores para los niveles de gris y para cada uno de éstos valores se recorrerá cada uno de los píxels de la imagen. Cada vez que el pixel analizado tenga el nivel de gris que se ha fijado, se suma 1 al contador asociado a este nivel de gris.

Tras terminar el barrido de la imagen para un nivel de gris determinado se guardará el valor del contador en un vector que tendrá 256 posiciones (una para cada nivel de gris). Se inicializará el contador a 0 y se pasará a analizar el siguiente nivel de gris, generando así el histograma de la imagen.

La función umbralizar también necesitará conocer cuál de los niveles de gris es el que se repite en mayor número de veces. Para ello calcularemos el máximo del vector obtenido y recorreremos cada nivel de gris hasta encontrar cuál de ellos se corresponde con este máximo.

Con estos valores ya podremos crear los umbrales y tendremos la imagen binaria lista para marcar.

9.2.4-FUNCIÓN MARCA

Con las imágenes binarias ya listas, el siguiente paso será macar la imagen original. Los píxels blancos de las imágenes binarias determinarán que píxels de la imagen original deben ser marcados. Para ello se ha creado la función marca, que se aplicará primero con la imagen original y el filtro para el prewitt horizontal, el resultado se marcará con la binaria del Prewitt vertical y así sucesivamente, obteniendo los bordes en todas las direcciones sobre la imagen original. La función marca presenta la siguiente forma:

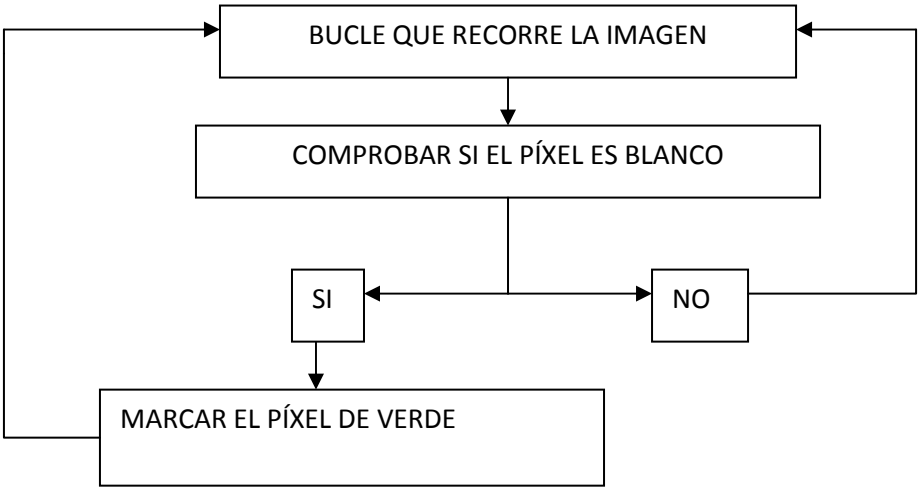


Fig. 9.2.4.a: Esquema de la función “marca”.

Necesitará como entrada dos matrices: la imagen a marcar y la binaria según la que se realizarán las marcas. Se barrerá la imagen binaria y se comprobará si el píxel en cuestión es de color blanco, en el caso de que se cumpla, al píxel correspondiente en la imagen a marcar, se le asignará un color verde (0,255,0 serán las componentes RGB).

Tras barrer toda la imagen, se tendrán marcados en verde todos los píxels blancos de la imagen binaria. El proceso se aplicará sobre las 4 imágenes binarias correspondientes a cada filtro Prewitt y así se terminará el primer procesado, obteniendo un resultado similar a la figura 9.2.4.b:

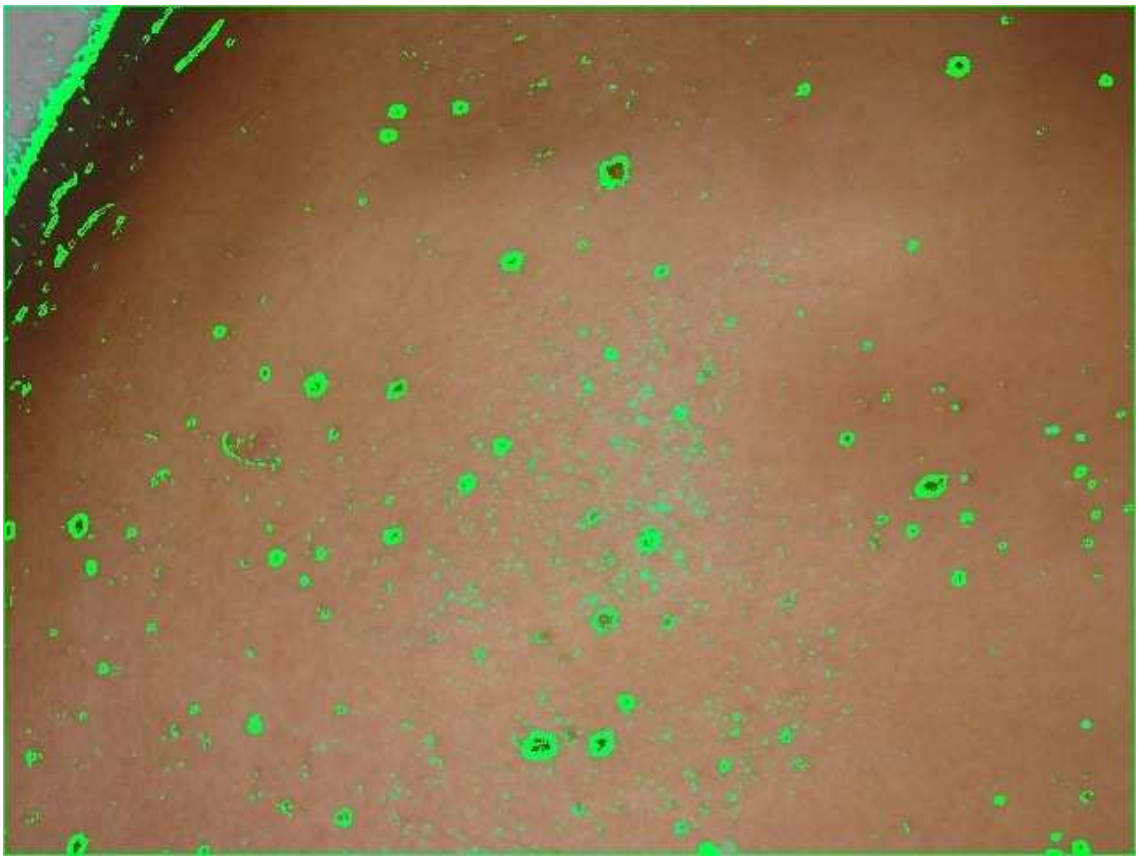


Fig. 9.2.4.b: Ejemplo tras terminar el primer procesado

En la imagen se pueden encontrar numerosos errores: hay marcas no correspondientes a lunares, los bordes de la imagen han sido marcados, los bordes correspondientes al fondo de la imagen en la parte superior derecha también, etc. Estos y otros errores tratarán de ser corregidos con un segundo procesado que será estudiado a fondo en el apartado siguiente.

9.3-SEGUNDO PROCESADO

Tras el primer procesado tendremos una imagen cuyas zonas de de mayor frecuencia (bordes) estarán marcadas. El problema radica en que, tal como se explicó anteriormente, no todos los bordes corresponden a las marcas que deseamos encontrar. Entre estas fronteras, se hallarán también marcas de vello, bordes de la propia piel respecto al fondo, textura de la piel, etc. Las características a tener en cuenta serán el color y forma de los lunares, el color de la propia piel y características correspondientes al vello. También hay que desechar algunas zonas por ser imposible que un lunar se encuentre allí (bordes de la imagen, fondo de la imagen,…). El esquema final para este segundo procesado quedará de la siguiente manera:

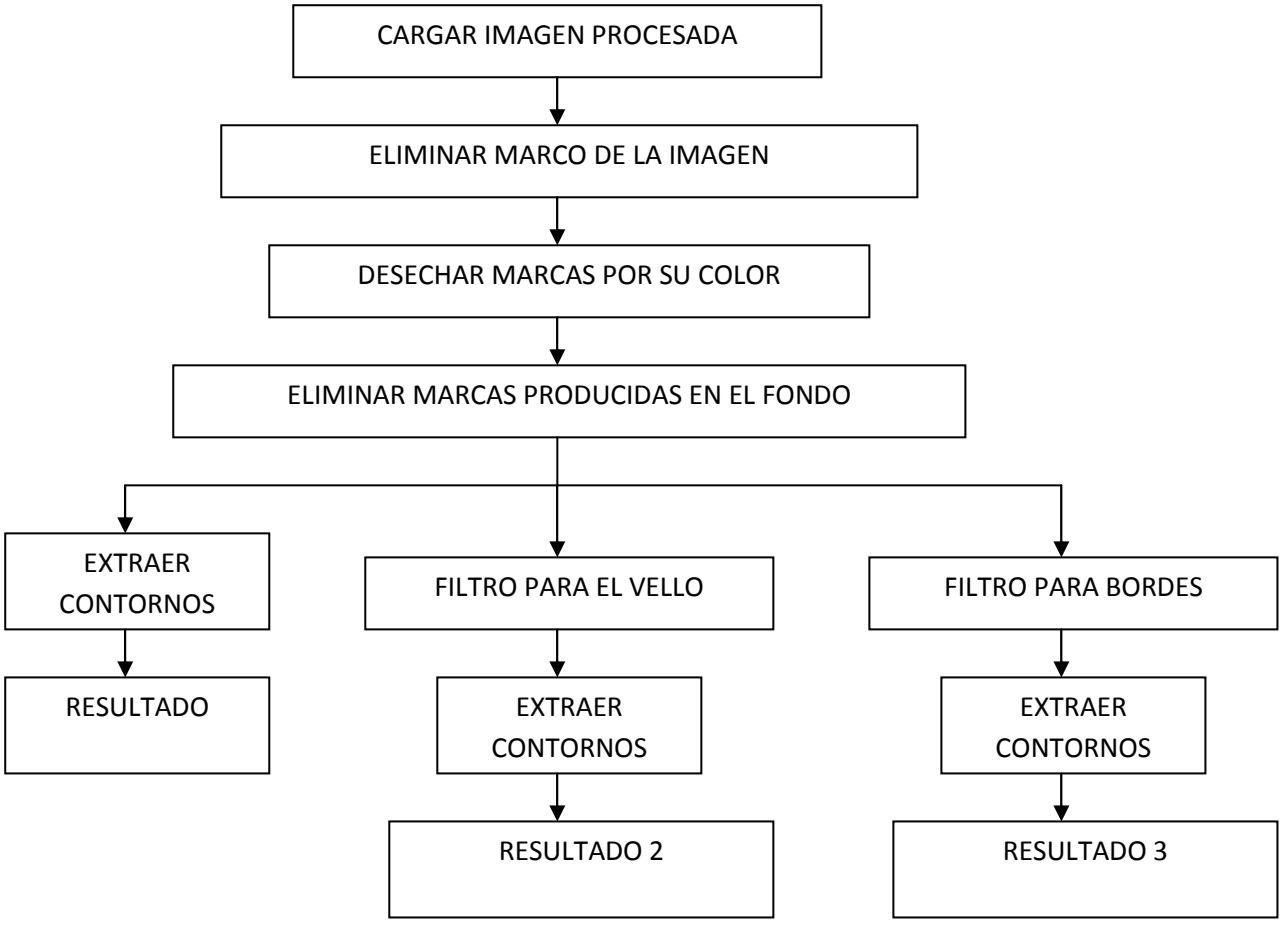


Fig. 9.3.a: Esquema general del segundo procesado.

Este segundo procesado requiere como entrada la imagen marcada resultado de aplicar el primer procesado para luego ir eliminando marcas que consideremos erróneas. Como se puede observar el programa dará tres resultados: en primer lugar el resultado sin aplicar ningún filtro, después el resultado de aplicar el filtro B y finalmente el resultado con el filtro P. Se decidió hacerlo de esta manera, porque aplicando filtros se pueden perder marcas correspondientes a lunares. Si en una imagen no tenemos vello ni bordes extraños (brazos, hombros, etc.) obtendremos mejores resultados si no aplicamos ninguno de los filtros.

En la primera parte del programa, delimitaremos un marco de 6 píxels y eliminaremos todas las marcas que coincidan con el marco. Así eliminaremos la marca producida por el filtro Prewitt en los límites de la imagen. El resto de filtros requieren una programación compleja, así que los explicaremos en apartados diferentes a continuación.

9.3.1- ELIMINAR MARCAS POR SU COLOR: FUNCIÓN BORRA

Esta función utiliza la información de color obtenida al hacer un muestreo de las marcas para desechar zonas que por su color no pueden ser lunares. Antes de entrar en detalle, observaremos los datos de color obtenidos para 20 lunares diferentes:

PUNTO	R	G	B	GRIS	H	S	V
1	173	143	117	144	41	19	98
2	74	55	38	56	20	55	91
3	173	142	124	146	12	61	57
4	98	70	48	72	348	13	95
5	110	74	48	77	16	56	73
6	189	157	134	160	9	72	40
7	153	122	101	125	7	46	54
8	188	152	128	156	52	9	98
9	164	132	109	135	22	45	82
10	170	140	116	142	36	53	60
11	135	98	82	105	20	23	72
12	167	126	98	130	25	22	70
13	84	46	27	52	13	28	60
14	150	109	81	113	23	71	36
15	169	128	98	132	18	39	53
16	100	58	34	64	27	56	55
17	146	101	70	106	36	53	60
18	159	123	111	131	41	19	98
19	88	54	44	62	27	56	55
20	122	84	61	89	20	28	50

Fig. 9.3.1.a: Características de color de 20 lunares

Si mostramos los valores de RGB en un gráfico se pueden apreciar mejor las pautas que sigue el color de los lunares:

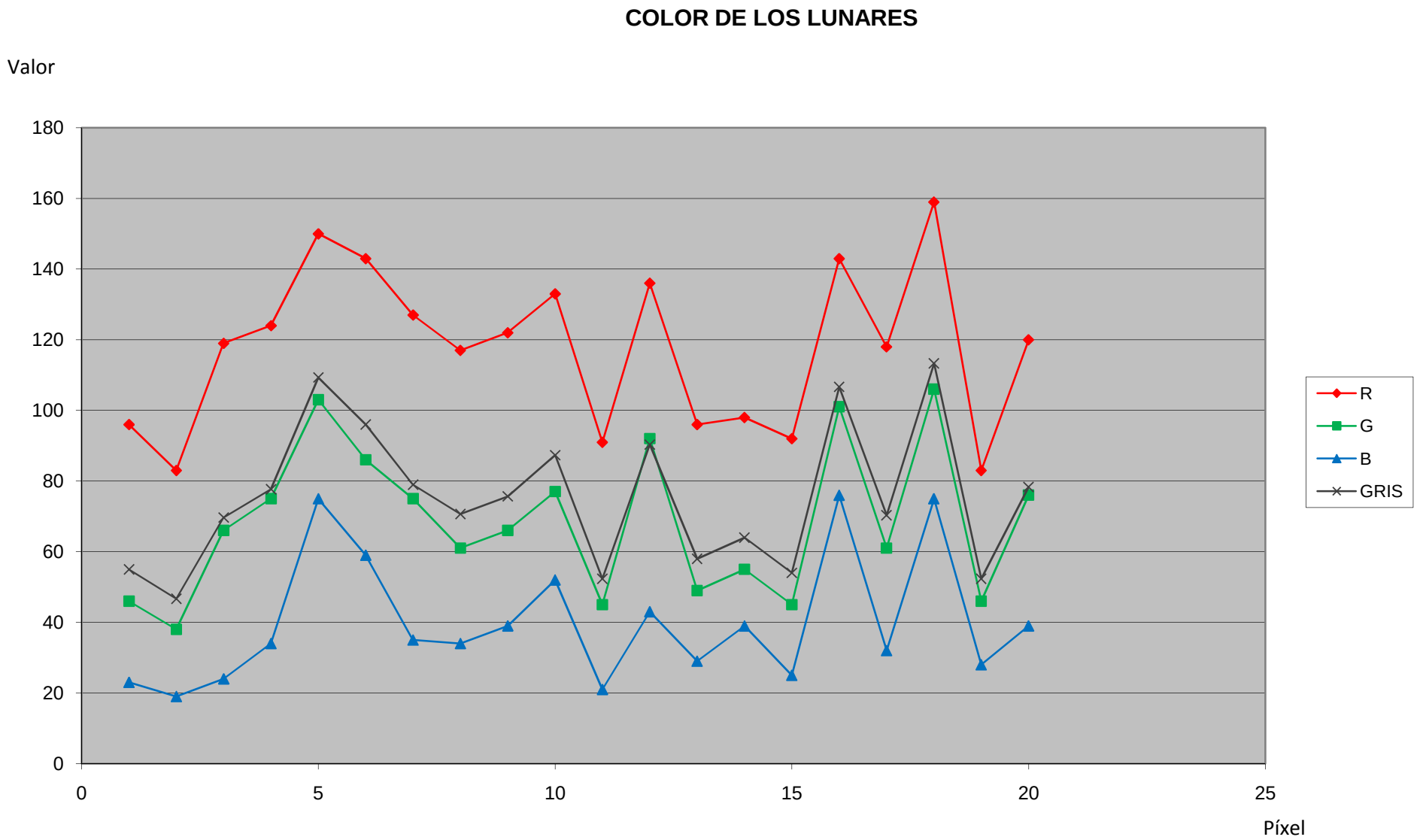


Fig. 9.3.1.b: gráfica con las componentes R, G, B y el nivel de gris de cada píxel muestreado en diferentes lunares.

El color de los lunares sigue unas pautas bien definidas: en todos los casos se cumplía que: los niveles de rojo se encontraban dentro de los rangos 83-159, los de verde entre 38-106, los de azul entre 19-76, los de gris entre 54-107, los de S entre 33-88, los de V entre 28-82. Se calculó que el nivel de rojo era superior al 80% de la suma de los niveles de verde y azul, se muestra en la columna final donde se ha aplicado la fórmula de restarle al nivel de rojo la suma de los valores de verde y azul multiplicada por un coeficiente de 0,8, como se observa, todos los resultados son positivos. En cuanto a los niveles de H, eran superiores a 354 o inferiores a 2, es decir tienen matices rojos (el matiz va de 0 a 354, pero es cíclico, es decir los valores muy altos o muy bajos corresponden a matices rojos). Con toda esta información se diseño el algoritmo de la figura 9.3.1.c:

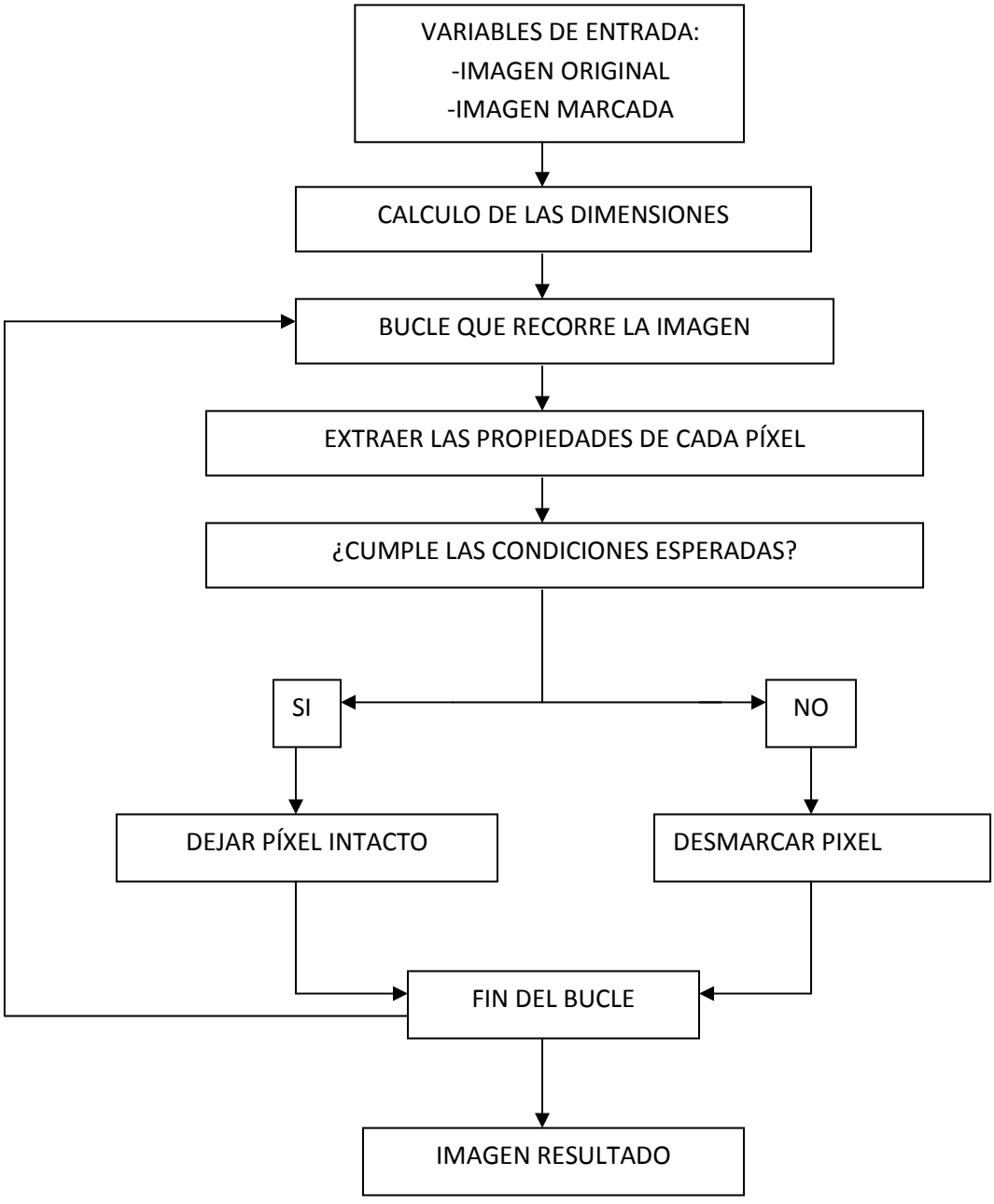


Fig. 9.3.1.c: Esquema de la función “Borra”.

El algoritmo necesitará la imagen marcada y la original como variables de entrada. En un primer paso calcula las dimensiones de las imágenes y realiza la transformación de la imagen original a HSV (matiz, saturación y brillo).

En los píxeles correspondientes de la imagen original, extraeremos los valores RGB y calcularemos el nivel de gris haciendo una media entre los tres. Extraeremos de la imagen en HSV los valores de matiz, saturación y brillo. Por último calcularemos el valor para la fórmula (f=R-0,8x(G+B)).

Con estos datos, podremos comprobar si el píxel cumple con las características de color explicadas anteriormente. En el caso de que se cumplan, el píxel se mantendrá de color verde para señalarlo como parte de un posible lunar. En el caso de que no se cumpla, daremos la marca por errónea y asignaremos los valores de la imagen original.

A continuación se presenta un ejemplo de una imagen marcada antes (figura 9.3.1.d) y después de aplicarle el filtro borra (figura 9.3.1.e). Se puede observar como varias de las marcas producidas por la textura de la piel y parte del fondo son eliminadas debido a que su color esta fuera de los límites establecidos para los colores de un lunar:

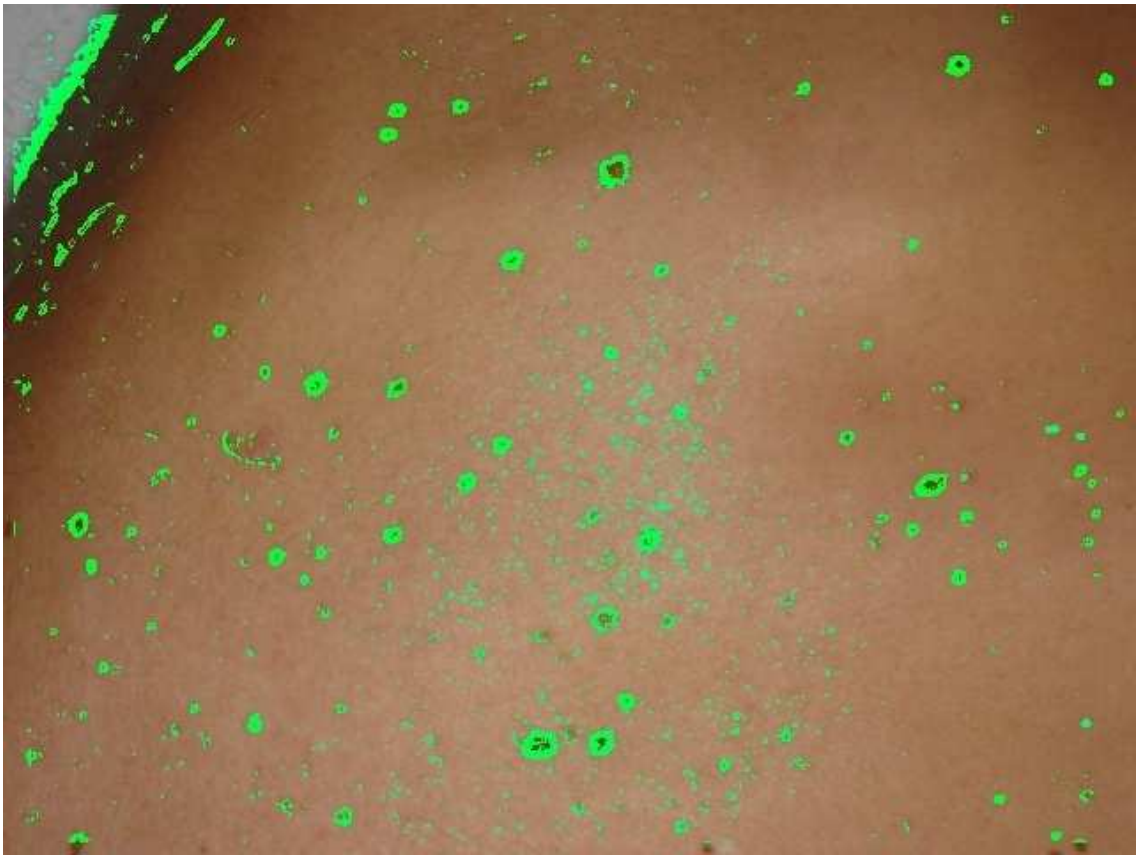


Fig. 9.3.1.d: Ejemplo antes de aplicarle la función “Borra”

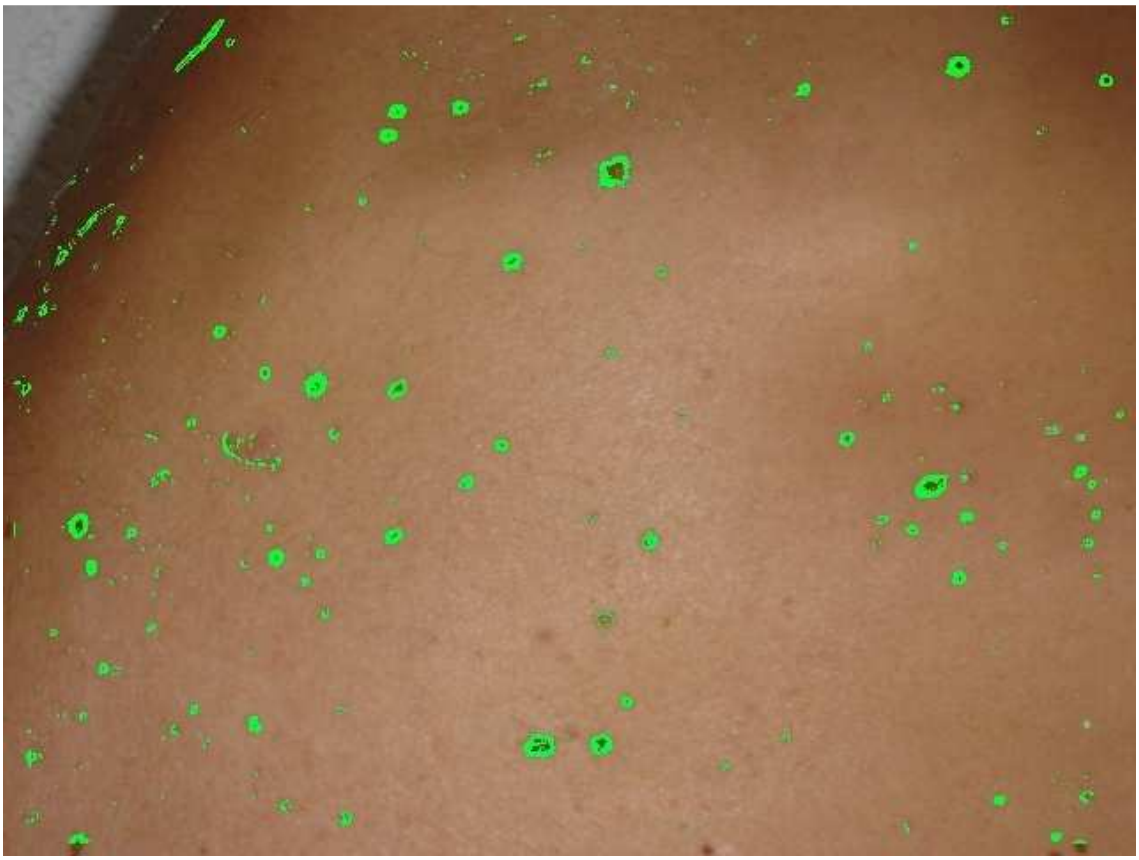


Fig. 9.3.1.e: Ejemplo después de aplicarle la función “Borra”.

9.3.2- FUNCIÓN PIEL

El objetivo de esta función es establecer una diferenciación entre lo que es la piel de la imagen y separarla totalmente del fondo. Con esto evitaremos marcar zonas del fondo que cumplan las características de color de los lunares Para delimitar la piel fue necesario establecer una serie de pautas de color. Realizamos un estudio del color de la piel con los siguientes resultados:

PUNTO	R	G	B	GRIS	H	S	V
1	173	143	117	144	41	19	98
2	74	55	38	56	20	55	91
3	173	142	124	146	12	61	57
4	98	70	48	72	348	13	95
5	110	74	48	77	16	56	73
6	189	157	134	160	9	72	40
7	153	122	101	125	7	46	54
8	188	152	128	156	52	9	98
9	164	132	109	135	22	45	82
10	170	140	116	142	36	53	60
11	135	98	82	105	20	23	72
12	167	126	98	130	25	22	70
13	84	46	27	52	13	28	60
14	150	109	81	113	23	71	36
15	169	128	98	132	18	39	53
16	100	58	34	64	27	56	55
17	146	101	70	106	36	53	60
18	159	123	111	131	41	19	98
19	88	54	44	62	27	56	55
20	122	84	61	89	20	28	50

Fig. 9.3.2.a: Características de color de 20 muestras de piel.

En la siguiente gráfica se observan mejor las pautas que sigue el color:

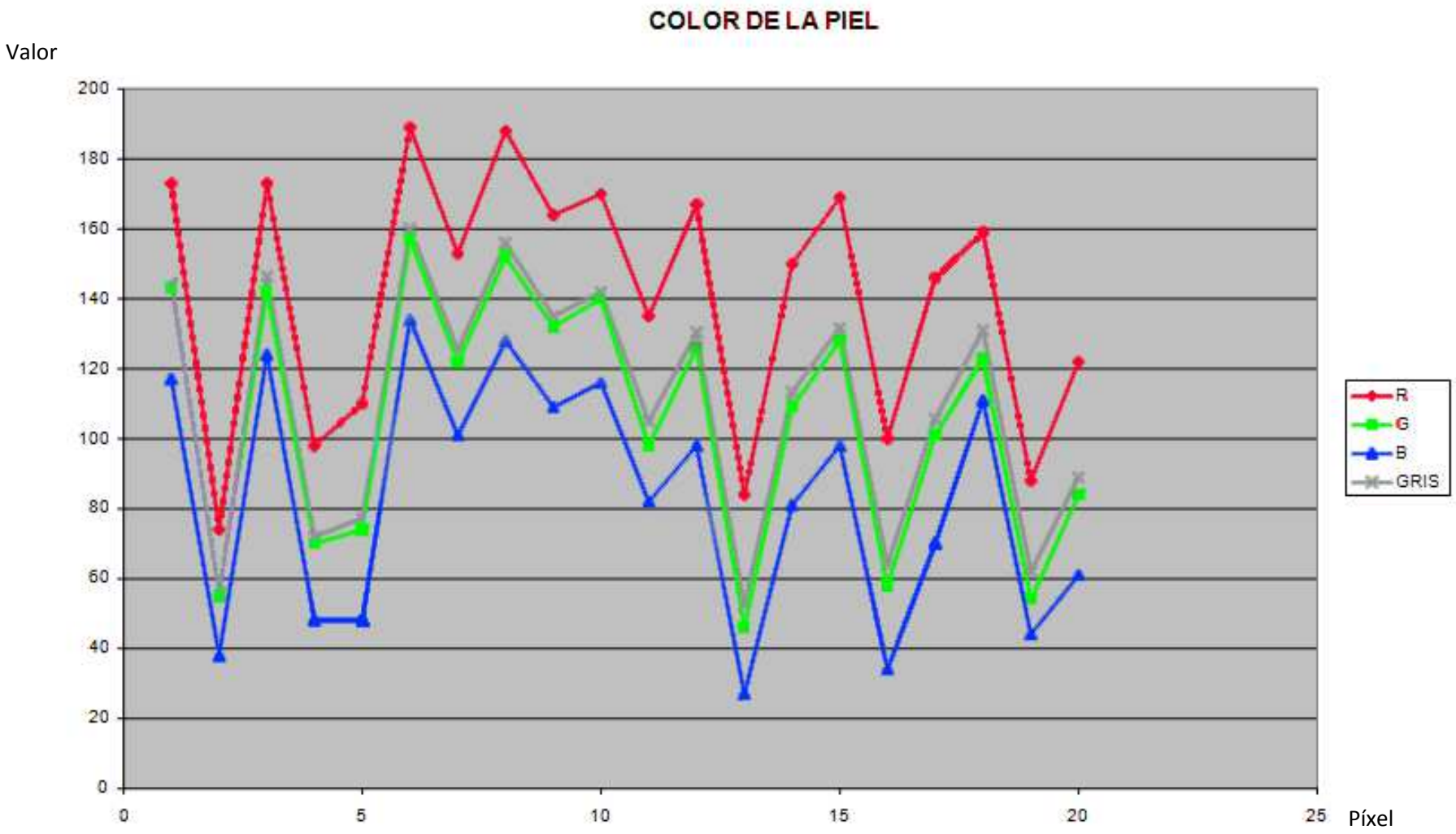


Fig. 9.3.2.b: Gráfica con las características de color para 20 muestras de la piel.

Con los datos obtenidos de nuestro estudio y los obtenidos empíricamente, llegamos a la conclusión de que la piel seguía los siguientes patrones de color: r>g & r>b & g+6>b & r>70 & g>40 & g<250 & b>25 & b<230 & gris>50 & gris<245 & s<75 & v>25.

Nuestro objetivo es que utilizando estos datos y con la imagen marcada como entrada, obtener imagen binaria en la que los píxeles de color blanco serán lo que hemos considerado como fondo de la imagen. Tras aplicar esta función, en "procesado2" se realizará un barrido, para asignar el valor de la imagen original sobre la imagen marcada en el caso de que el píxel en cuestión sea blanco (eliminando las marcas en las zonas consideradas como fondo). Para ello, barreremos la imagen desechando los píxels que por sus características de color no pueden pertenecer a la piel. Seguiremos el procedimiento esquematizado en la figura 9.3.2.c:

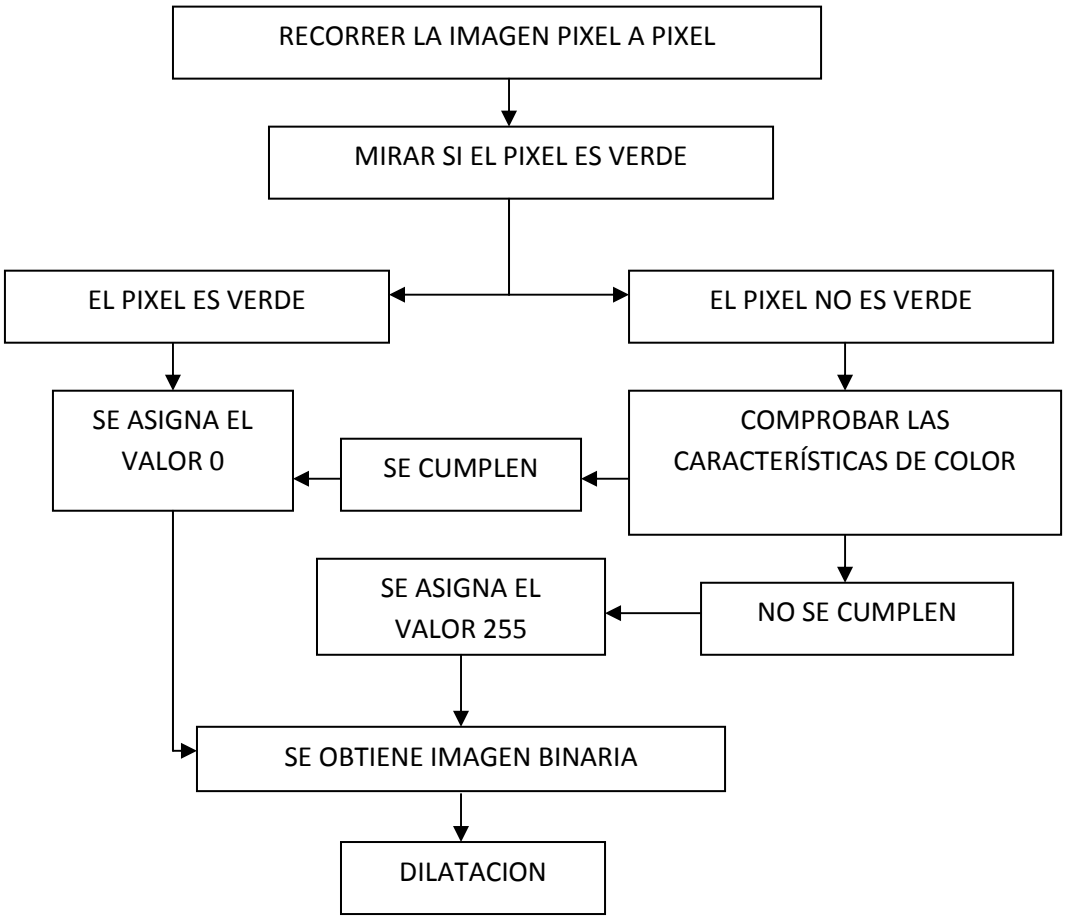


Fig. 9.3.2.c: Esquema del algoritmo utilizado para desechar marcas del fondo de la imagen.

En primer lugar, comprobaremos si el píxel a examinar es de color verde. En el caso de que sea verde, será un píxel perteneciente a las marcas, así que le asignaremos el valor 0 directamente y nos ahorraremos realizar varios cálculos y comprobaciones, agilizando la ejecución del programa. Si el píxel no es verde calcularemos y comprobaremos las características, asignando el valor 255 en el caso de que no se cumplan los patrones de color.

Para obtener una imagen binaria que encaje de mejor manera con el fondo esperado, el siguiente paso será barrer la imagen desde otras direcciones, es decir empezando de las otras esquinas de la imagen. Seguidamente se realizará una dilatación para desechar los píxeles de los bordes, evitando que se marquen píxeles correspondientes al borde pero con características de color de la piel.

Esta técnica presenta un inconveniente: zonas que no tienen color de piel pero que se encuentran dentro de la piel son consideradas como fondo, con el problema de que al realizar la dilatación, podemos eliminar marcas que sí que son interesantes. Para evitarlo añadiremos otra técnica. Recorreremos la imagen desde los bordes, añadiendo los píxeles que no cumplan las características de color y que hayan sido considerados como fondo en la etapa anterior. Pararemos de añadir píxels cuando lleguemos a la frontera con la piel. Esta técnica se basa en que el fondo estará siempre pegado a uno de los bordes de la imagen, es decir no va a haber parte del fondo rodeado completamente de piel. El esquema de funcionamiento será el mostrado en la figura 9.3.2.d, repitiéndose empezando por cada una de las esquinas.

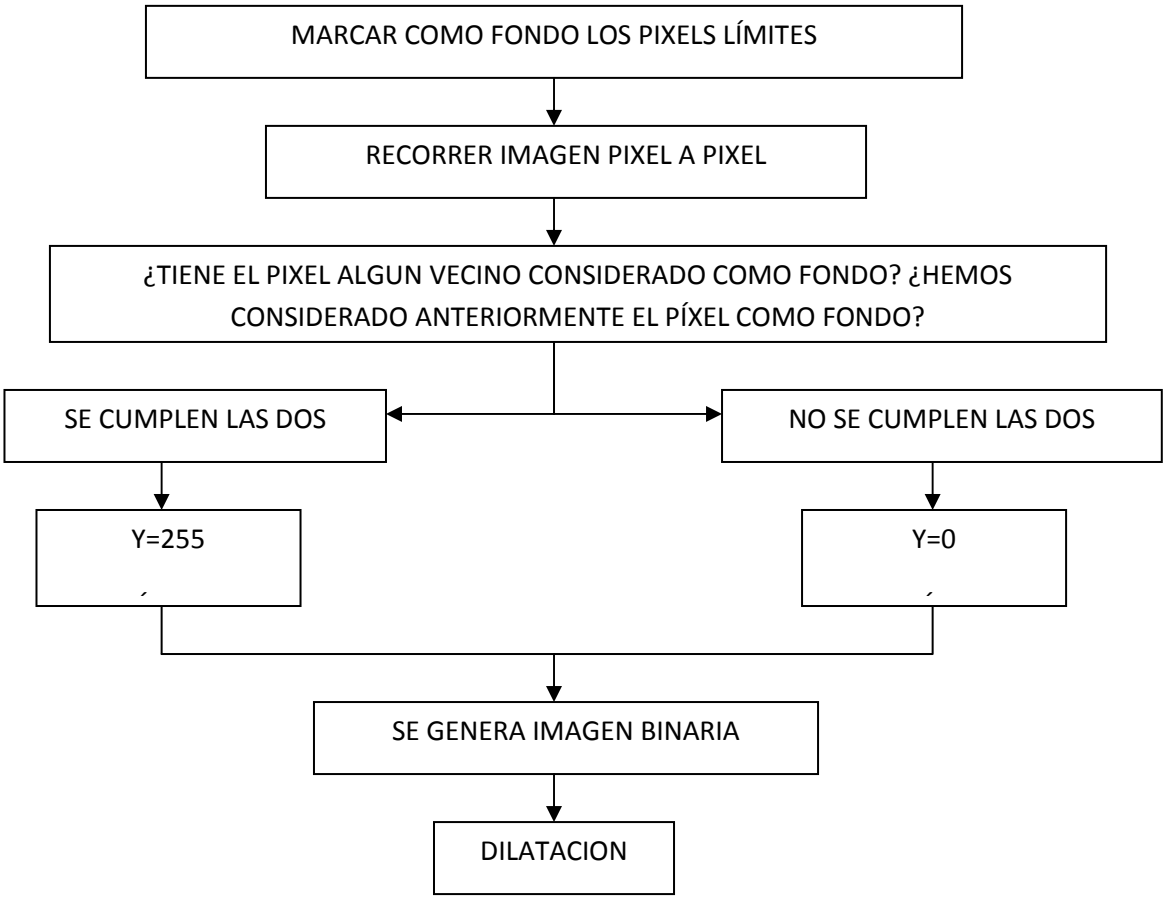


Fig. 9.3.2.d: Corrección de la primera estimación de fondo.

Empezaremos marcando como fondo los píxels que delimitan la imagen. Recorreremos los píxels empezando por una de las esquinas, añadiendo como fondo los píxels que tengan un vecino blanco y que además hayan sido considerados como fondo en la primera estimación. Así incluiremos píxels que cumplan con las características de color del fondo y que no estén rodeados de piel, como era nuestro objetivo.

Repetiremos este paso barriendo la imagen empezando por cada una de las esquinas y siguiendo un sentido diferente. Generaremos 4 imágenes binarias diferentes que fusionaremos dando lugar a la imagen binaria final. Para finalizar realizaremos una dilatación para desechar marcas en los bordes de la piel con el fondo. Obtendremos una imagen binaria que se utilizará en la función “procesado2” para desechar el fondo. A continuación tendremos una imagen original, la imagen binaria resultado de aplicar la función “piel” y cómo se eliminan las marcas tras utilizar esta función:

Fig. 9.3.2.e: Imagen original.

Fig. 9.3.2.f: Estimación del fondo.

Fig.: 9.3.2.g: Imagen marcada.

Fig. 9.3.2.h: Resultado tras eliminar el fondo.

Los resultados son lógicos, la parte correspondiente al fondo es de forma grisácea tal y como se ha marcado en la imagen binaria. Como se ve en las dos imágenes finales, gran parte de las marcas correspondiente a los límites de la piel con el fondo (hombros, costado,…) han sido eliminadas.

9.3.3- FILTRO PARA VELLO (FILTRO P)

La mayor problemática a la hora de desarrollar los algoritmos de detección de lunares, ha estado sin duda en las imágenes que presentaban vello. El vello presenta colores similares a los de los lunares, además de que las zonas con vello son zonas de alta frecuencia, que los filtros Prewitt utilizados en el primer procesado marcarán como zonas de borde. A continuación presentamos una imagen original (figura 9.3.3.a) con vello y el resultado de aplicar un primer procesado (figura 9.3.3.b), mostrando la gran cantidad de errores que se comete:

Fig. 9.3.3.a: Imagen original con vello.

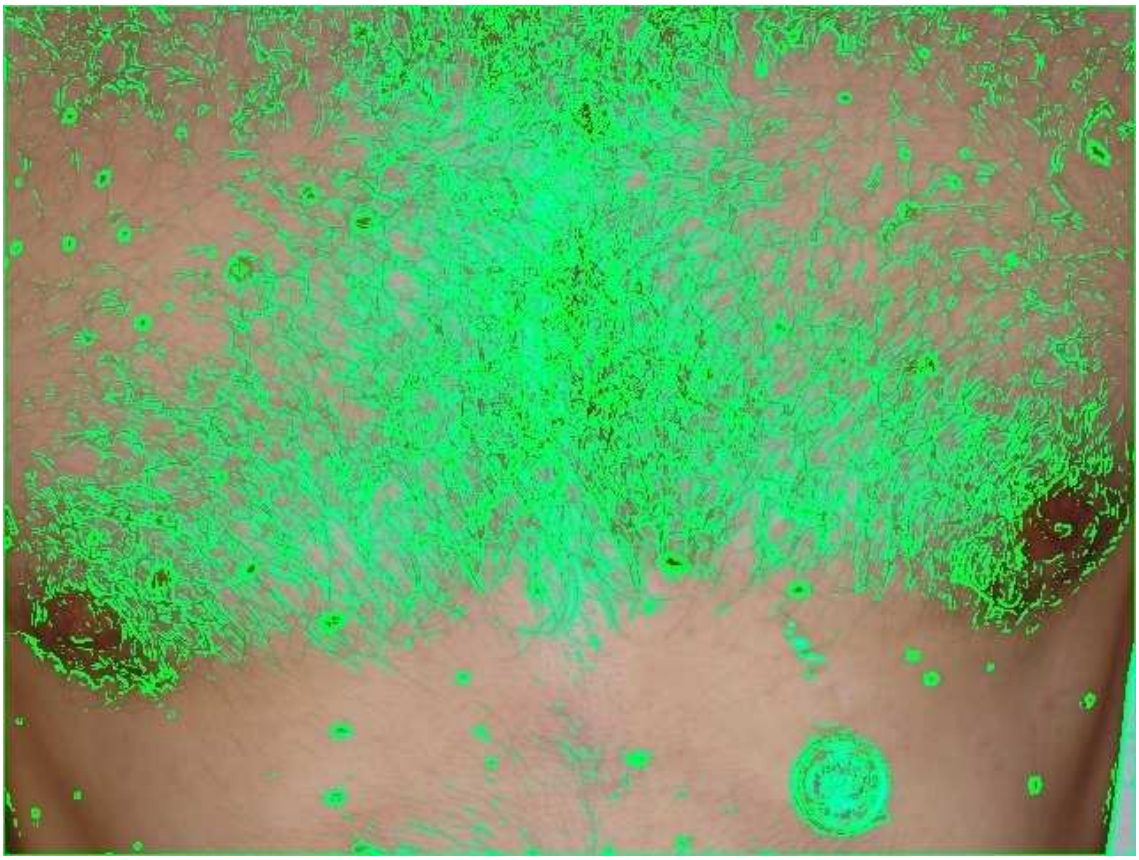


Fig. 9.3.3.b: Resultado tras el primer procesado.

La dificultad es tal, que no se ha conseguido llegar a un resultado óptimo (pero sí aceptable) cuando se utiliza el filtro implementado. El filtro P no consigue eliminar todas las marcas correspondientes al vello y en ocasiones elimina otras marcas que sí que son interesantes. Por ello se decidió dar un resultado aparte, en el que se muestra la imagen filtrada con el algoritmo que describiremos a continuación.

El algoritmo a analizar, utiliza principalmente las dos características más significativas del vello: es muy fino y es más oscuro que la piel. En primer lugar utilizaremos filtro Laplaciano de 5x5, cuya máscara podemos observar en la figura 9.3.3.c:

1	1	1	1	1
1	1	1	1	1
1	1	-24	1	1
1	1	1	1	1
1	1	1	1	1

Fig:9.3.3.c: Máscara del filtro Laplaciano utilizado.

Para implementar esta máscara, definiremos una variable “suma” que será el resultado de multiplicar el valor del píxel central por -24 y sumarle el valor del resto de píxels incluidos en la máscara. Comprobaremos cada píxel si el filtrado supera un umbral establecido en 150 (los valores negativos los consideraremos como “0”). Si lo hace consideraremos ese píxel como parte del vello, después aplicaremos otras técnicas para obtener un resultado óptimo.

Esta máscara permitirá no tener en cuenta como parte de vello, las zonas uniformes (recordemos que las zonas de vello son zonas de alta frecuencia y por tanto no uniformes), ya que el valor del píxel central multiplicado por -24 será contrarrestado por el valor de los demás píxels que será similar al del centro.

Por el contrario los píxeles que serán considerados vello, serán píxeles con niveles altos de gris, que tengan cerca niveles bajos de gris. No se marcarán píxeles con niveles bajos de gris, ya que el valor del píxel central será bajo. En resumidas cuentas, se marcarán bordes entre zonas claras y oscuras de la imagen. Pasaremos entonces a realizar una dilatación con una máscara de 5x5, lo cual permitirá marcar los píxeles oscuros que antes no habían sido tenidos en cuenta, y también permitirá poder desechar marcas finas (pertenecientes al vello) manteniendo las gruesas (pertenecientes a lunares).

En la práctica, las técnicas utilizadas hasta ahora generarán una “gran masa” en zonas de mucho vello, en lugar de marcar pelo a pelo. Esto puede provocar que lunares cercanas a la zona de vello sean consideradas vello. Para corregirlo de la mejor manera posible, se utilizará una técnica consistente en que si las marcas son mayores de un tamaño dado, se considerarán parte de la masa considerada como vello. Si las marcas son más pequeñas, se considerarán lunares. El esquema del algoritmo quedará como se muestra en la figura 9.3.3.d:

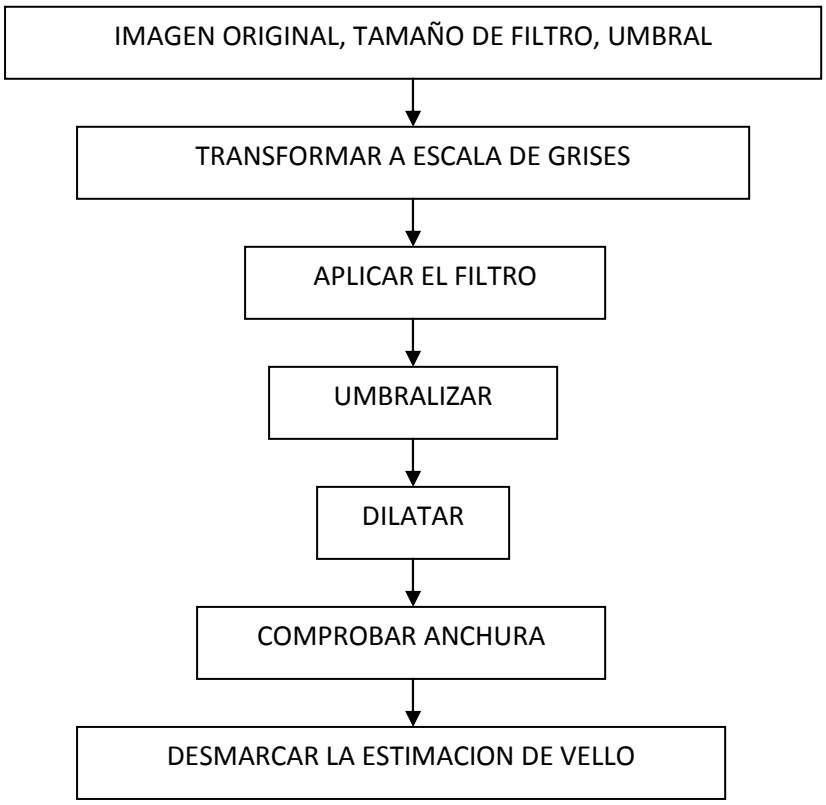


Fig. 9.3.3.d: Esquema para la estimación de vello.

El algoritmo requerirá 3 entradas: la imagen original, el tamaño de la máscara y el umbral para la máscara. Los resultados óptimos se consiguieron con una máscara de 5x5 y con un umbral de 150, por ello la llamada a esta función desde el programa principal se realiza directamente con estos valores.

En primer lugar aplicaremos el filtro de tamaño indicado. Si el filtro es de tamaño K, se tendrán en cuenta K al cuadrado píxels rodeando al píxel central. Se tendrán que utilizar los píxels dentro de un radio de K/2 en torno al central. Se multiplicara el píxel central por K al cuadrado menos 1 y se le restará el valor de todos los píxels que entren dentro de la máscara. Aplicando la máscara a todos píxels de la imagen obtendremos la imagen en escala de grises correspondiente al filtro. Un ejemplo del resultado se puede ver en las siguientes imágenes original (9.3.3.e) y filtrada (9.3.3.f):

Fig. 9.3.3.e: Imagen original.

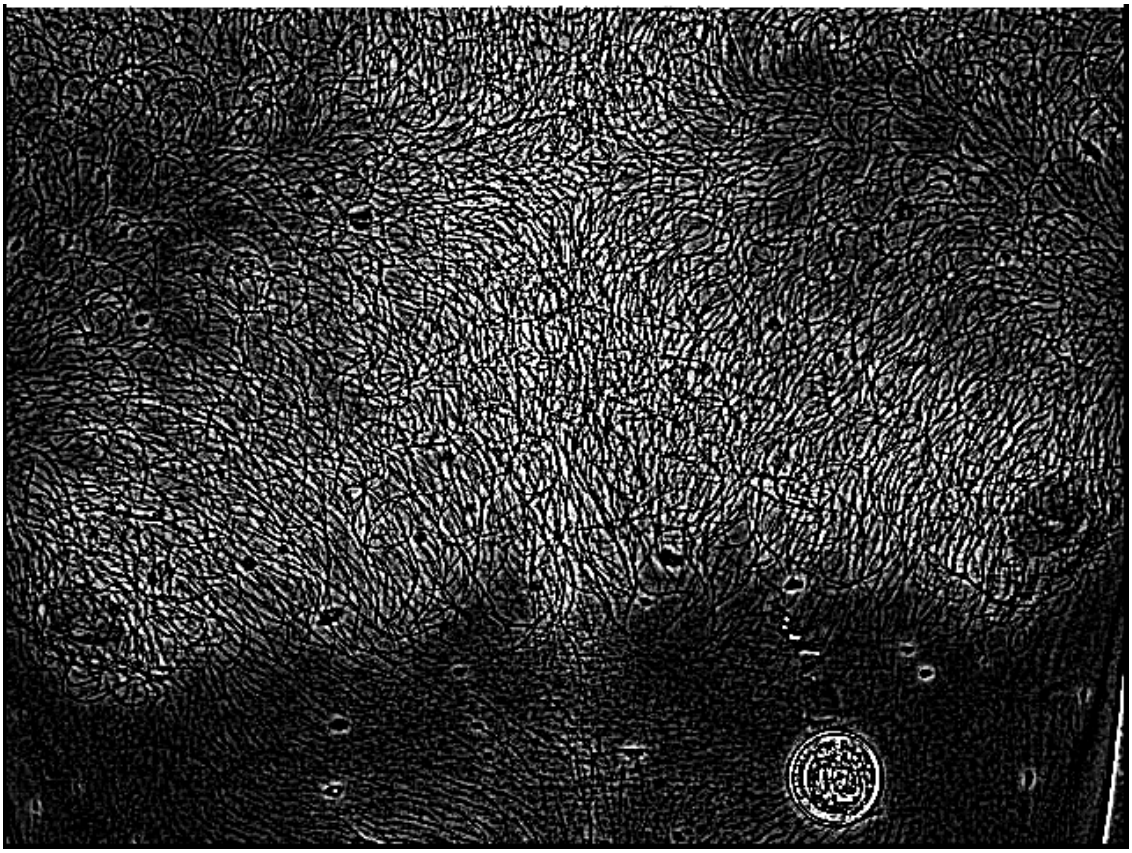


Fig. 9.3.3.f: Resultado del filtrado Laplaciano.

El siguiente paso será umbralizar la imagen generando una binaria que haga una primera estimación de lo que es el vello. Si el nivel de gris obtenido para cada píxel es superior al umbral se le asignara el valor 255, si no lo supera, el 0. El resultado para el ejemplo anterior será el siguiente:

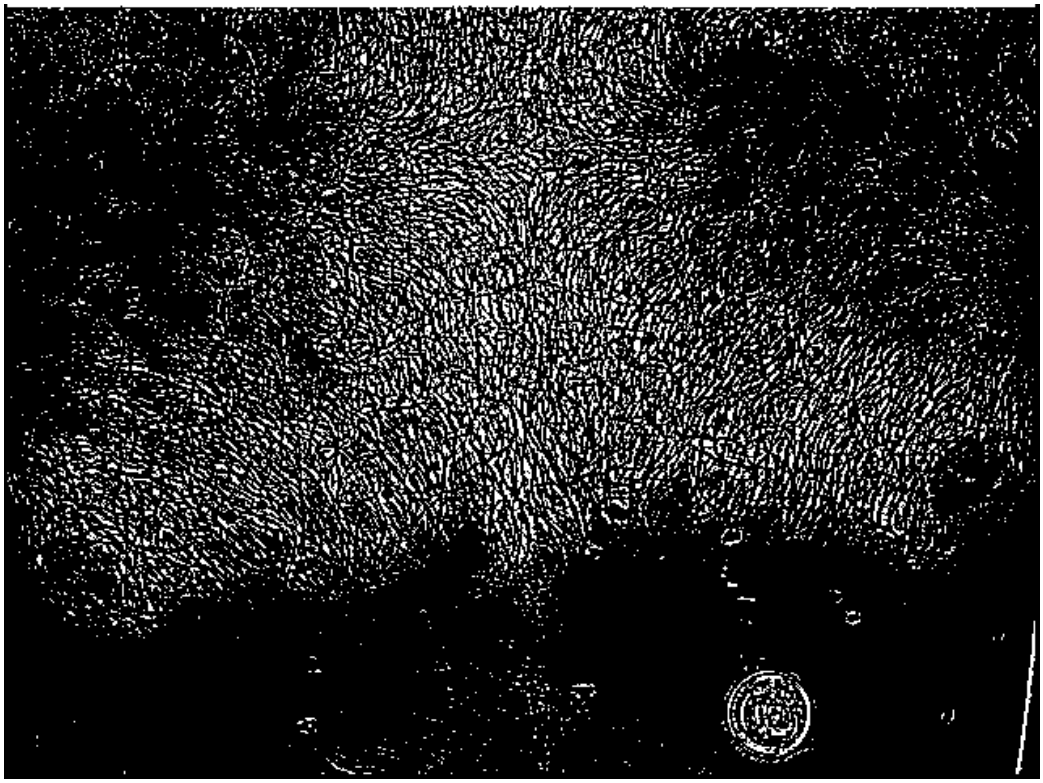


Fig. 9.3.3.g: Umbralización del Laplaciano.

A continuación, se realizará la dilatación como se ha explicado antes. En la figura 9.3.3.h se puede ver la masa generada por las zonas de vello, que luego tendremos que aplicar la técnica para desechar marcas menores de 40 píxels para no descartar lunares considerándolos como vello.

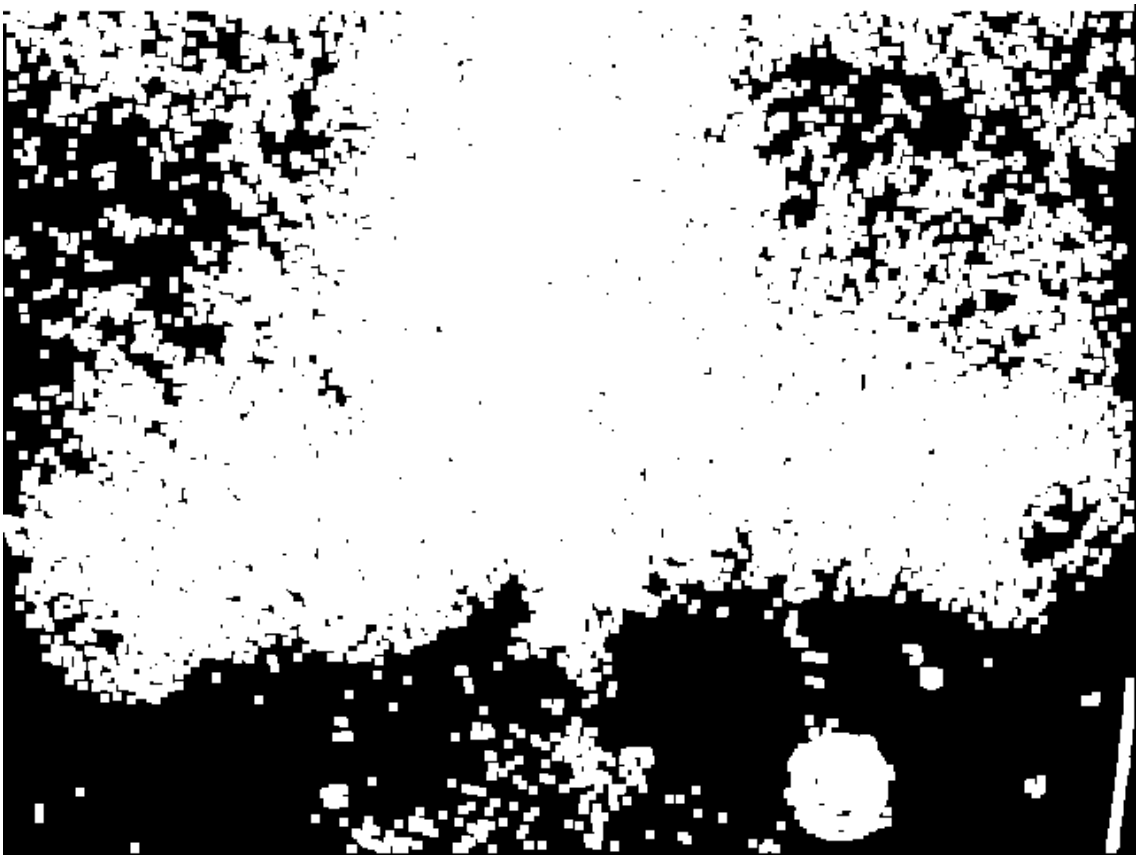


Fig. 9.3.3.h: Masa obtenida tras la dilatación.

Es el momento de utilizar la propiedad de que el vello forma una masa, diferenciándolo de las marcas más pequeñas, que sí que serán las correspondientes a lunares. En último paso de la función para el filtro P, recorreremos la imagen hasta dar con un píxel en blanco. En ese momento se establecerá un contador a 0 y recorrerá el resto de la fila sumando 1 al contador cada vez que tengamos un píxel blanco.

Cuando se encuentre un píxel negro o se llegue al final de la fila, pararemos de sumar. Entonces aplicaremos la condición de que si la suma es mayor de 40 píxels (todos los lunares son menores de este tamaño) consideraremos esta marca como parte de el vello, mientras que, si es menor la consideraremos un lunar.

Realizaremos un barrido de izquierda a derecha. En un principio se pensó en hacer un barrido en las cuatro direcciones, pero el algoritmo se complicaba mucho, resultaba mucho más lento y los resultados eran similares. La aproximación final para nuestro ejemplo será la mostrada en la figura 9.3.3.i:

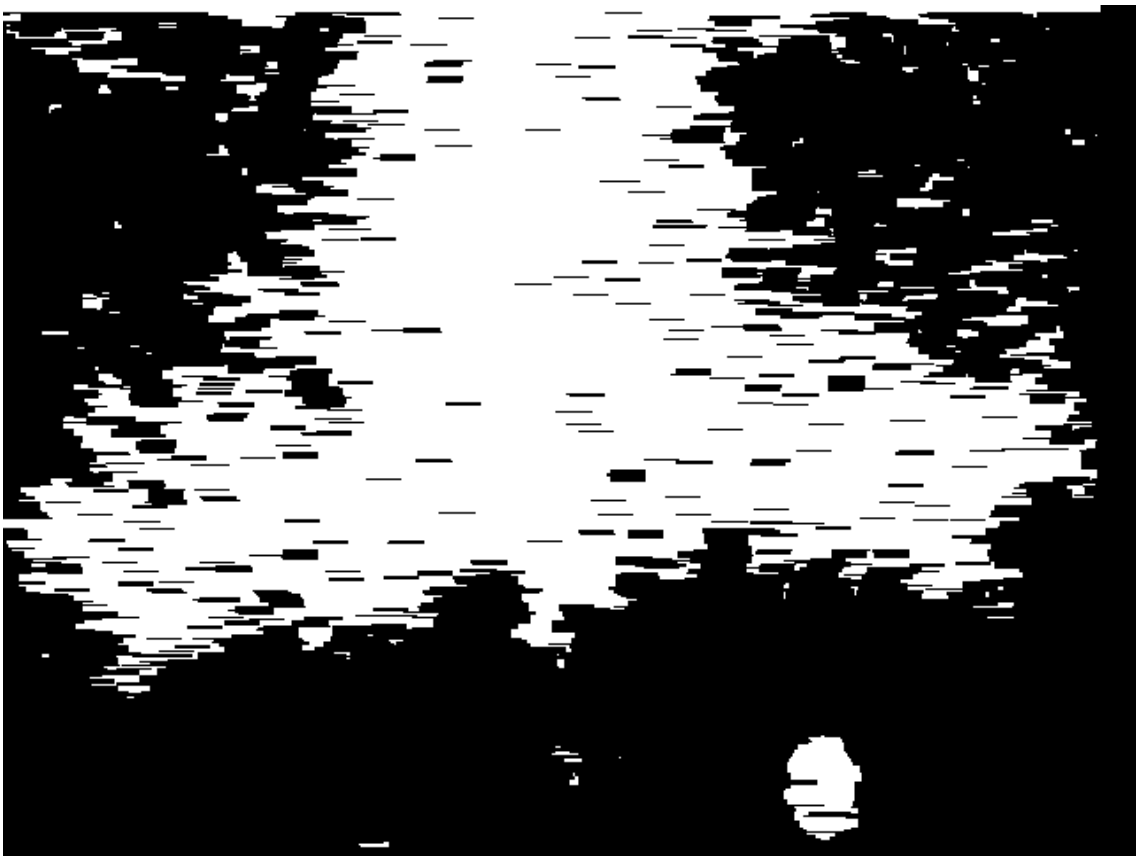


Fig. 9.3.3.i: Estimación final del vello.

Para terminar tendremos que desechar los píxels blancos de esta imagen binaria de la imagen marcada, permitiendo desechar gran parte de las marcas producidas por el vello. A continuación se muestra la imagen marcada (figura 9.3.3.j) y el resultado tras aplicar el filtro P (figura 9.3.3.k):

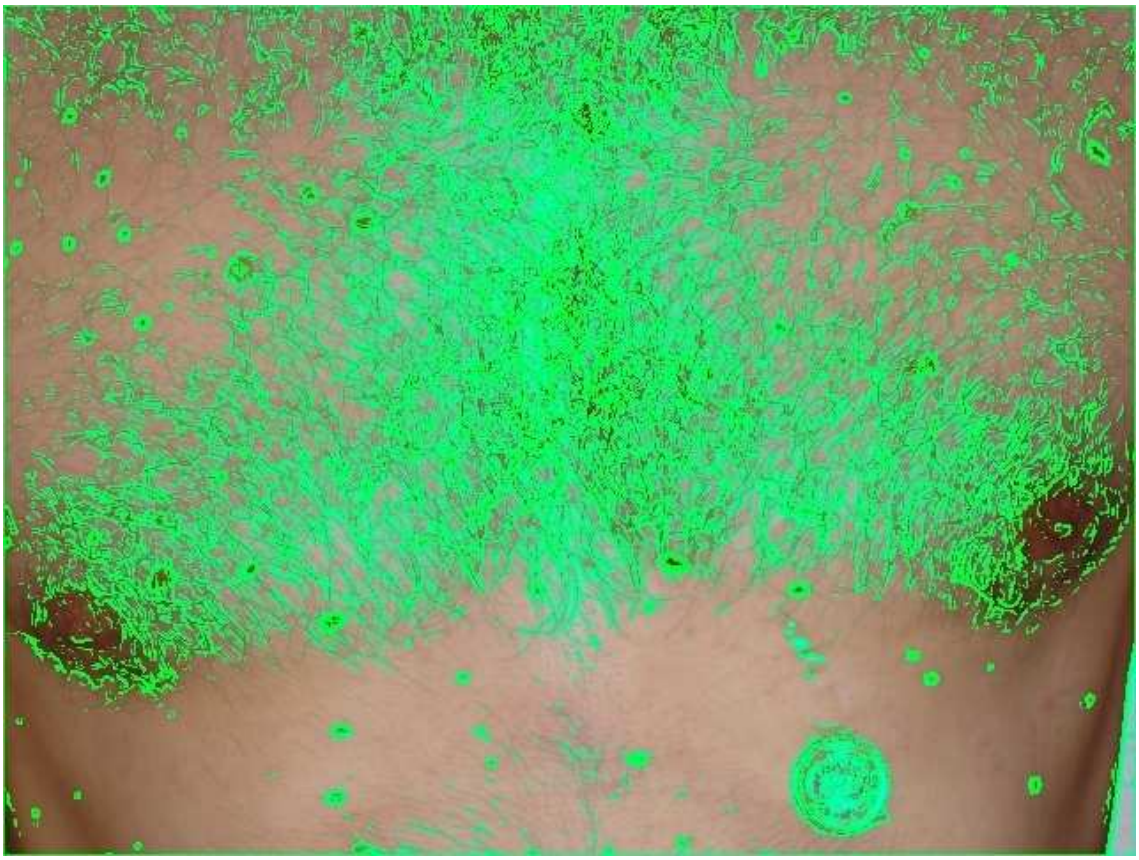


Fig. 9.3.3.j: Ejemplo antes de aplicar el filtro P.

Fig. 9.3.3.k: Ejemplo tras aplicar el filtro P.

9.3.4-EXTRACCIÓN DE CONTORNOS

El resultado de todos los procesos realizados hasta ahora son dos imágenes cuyas zonas consideradas como lunares están marcadas de color verde. Aunque están marcados los bordes, cada marca ocupa gran parte de la superficie del propio lunar, perjudicando en el análisis de resultados, como se puede ver en el siguiente ejemplo:

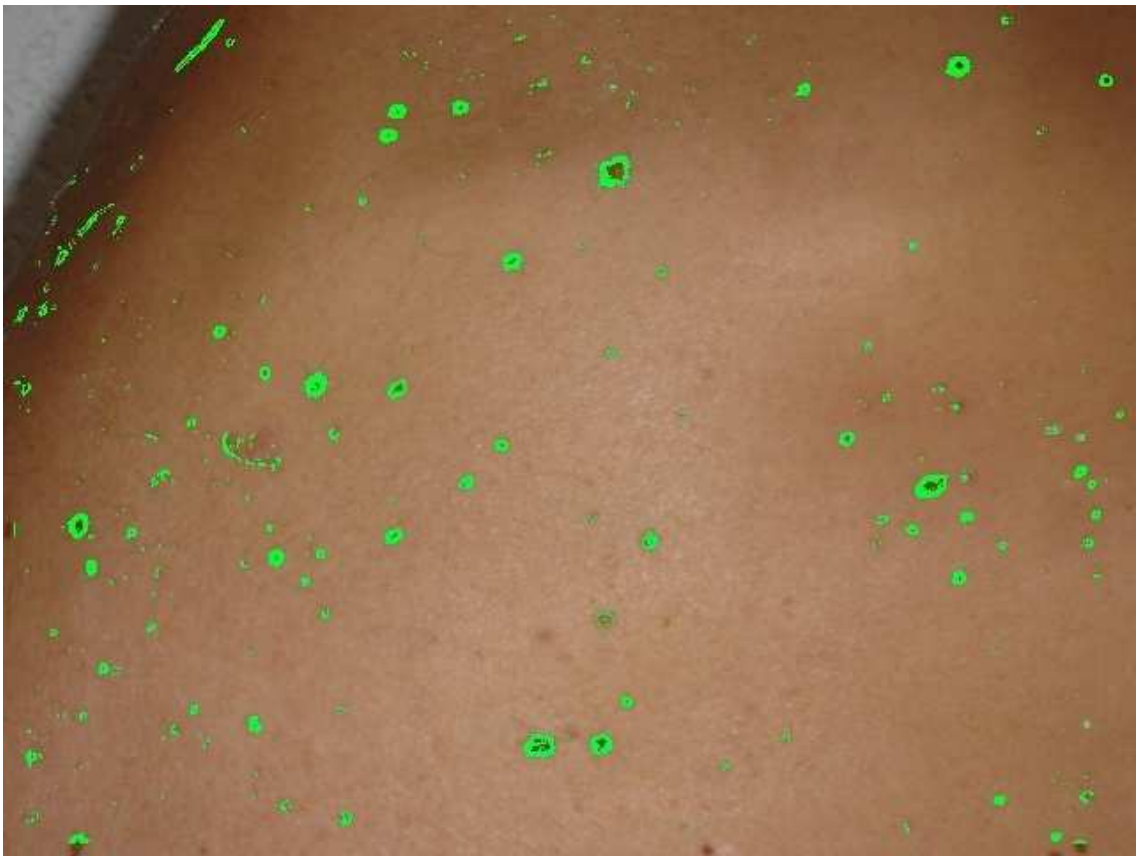


Fig. 9.3.4.a: Resultado sin poder analizar el interior de las detecciones.

Para obtener resultados óptimos para ser analizados y para reconocer el lunar, optaremos por una extracción de contornos, que permitirá marcar tan sólo los bordes de los lunares dejando el interior listo para ser analizado. El esquema del algoritmo será el mostrado en la figura 9.3.4.b:

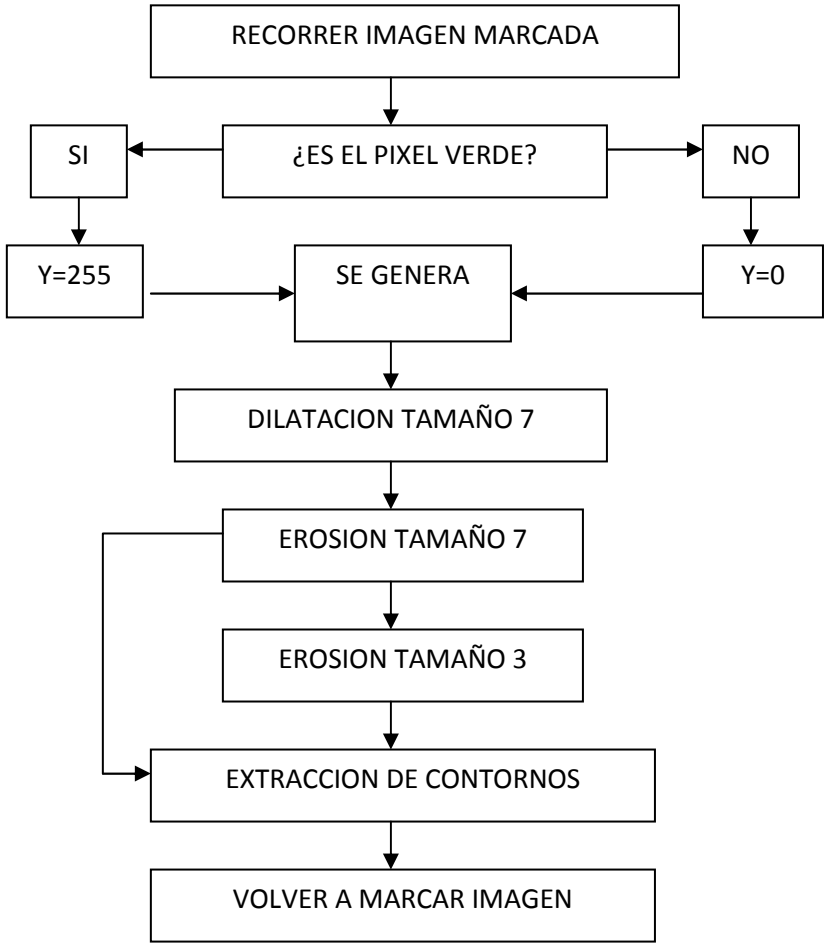


Fig. 9.3.4.b: Esquema de extracción de contornos.

Como entrada del algoritmo, precisaremos de una imagen marcada como las obtenidas anteriormente. En la primera parte del procesado, barreremos la imagen buscando los píxels pertenecientes a las marcas, generando una imagen binaria cuyos píxels blancos corresponden a las marcas y los negros al resto de la imagen. El resultado para nuestro ejemplo será el siguiente:

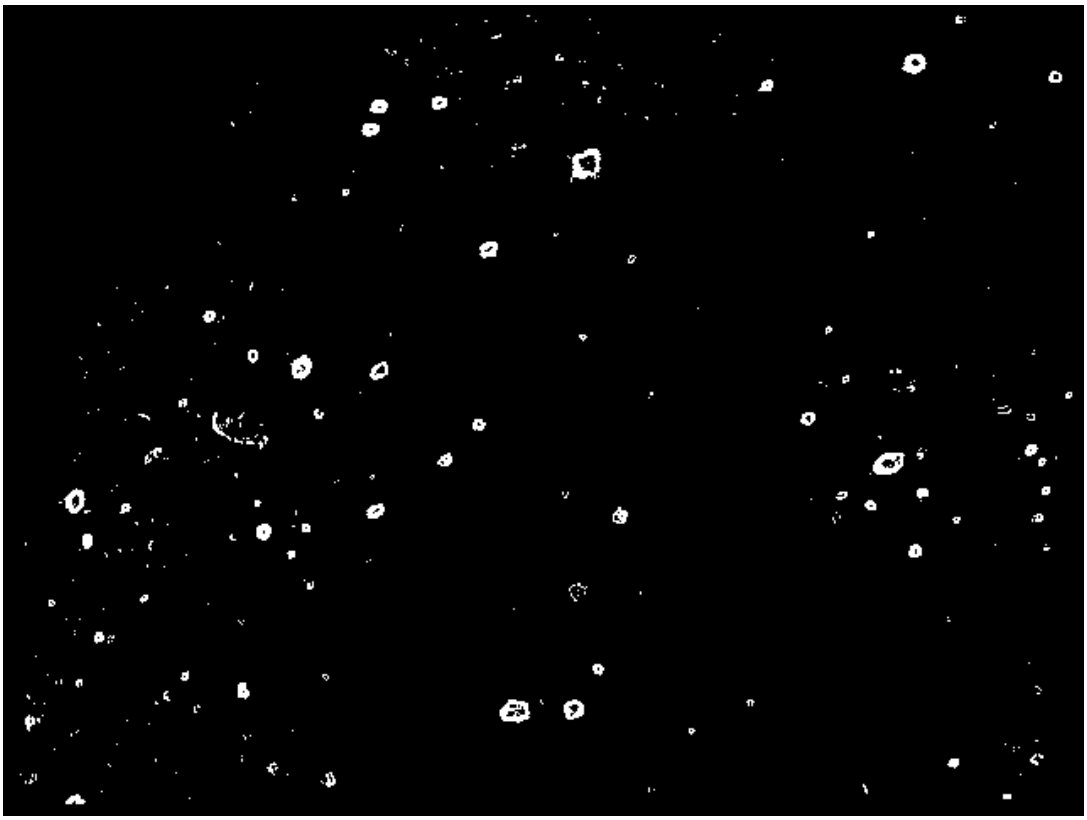


Fig. 9.3.4.c: Imagen binaria con las marcas obtenidas hasta ahora.

El siguiente paso será realizar una dilatación de tamaño 7, seguida de una erosión del mismo tamaño (es decir, realizaremos un cierre). Con esto, conseguimos rellenar el interior de los lunares (si no la extracción de contornos se aplicaría sobre los bordes del lunar y no sobre el propio lunar) dejando el tamaño de las marcas intacto. El resultado será el siguiente:

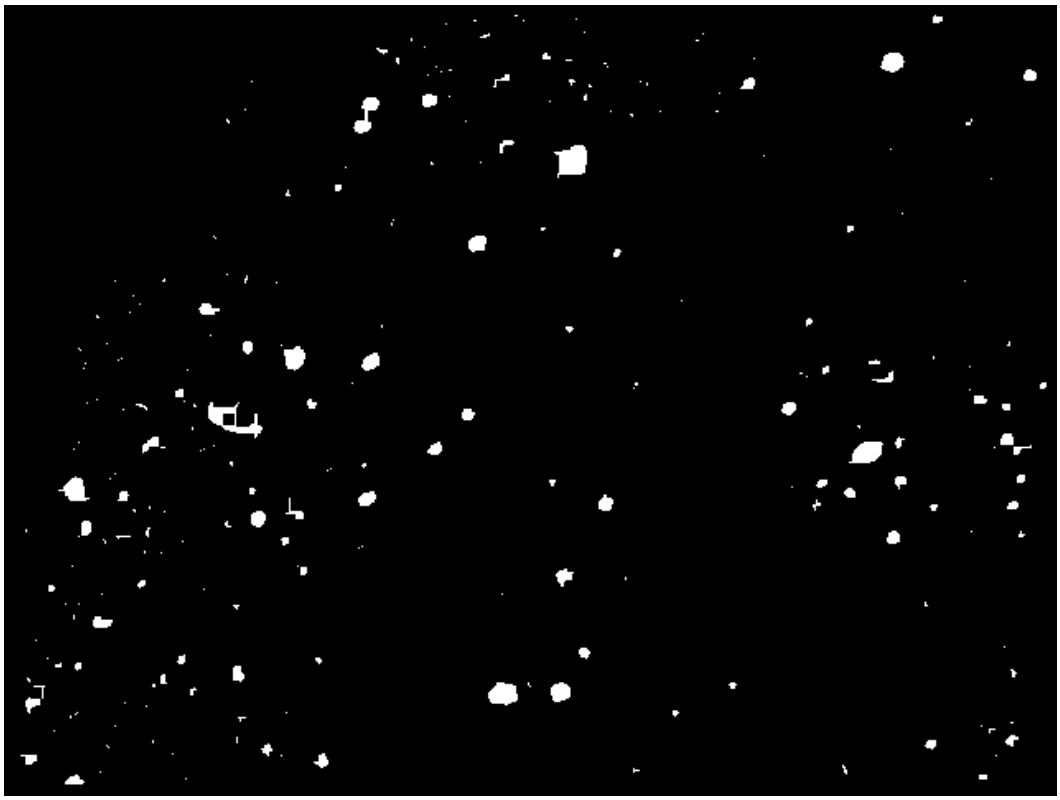


Fig. 9.3.4.d: Dilatación de las marcas.

En este momento, se realizará una erosión de tamaño 3 y se guardará en otra copia diferente. Tendremos ahora la imagen que cubre todo el lunar y la copia, que cubre todo el lunar excepto una erosión. Si recordamos la teoría de morfología, si a una marca le restamos su erosión, tendremos una forma de obtener su contorno.

Para ello, recorreremos los píxels de la imagen, dándoles un valor de 255 si el píxel está incluido en la imagen completa, pero no en la erosionada. Además asignaremos el valor 0 en el caso de que no se cumpla la condición. Podremos volver a marcar nuevamente la imagen, señalando tan sólo los contornos. A continuación podemos ver la imagen de entrada (figura 9.3.4.e) y el resultado tras la extracción de contornos (9.3.4.f). Se observa claramente que después de aplicar la función, se puede apreciar correctamente el interior de los lunares para su análisis:

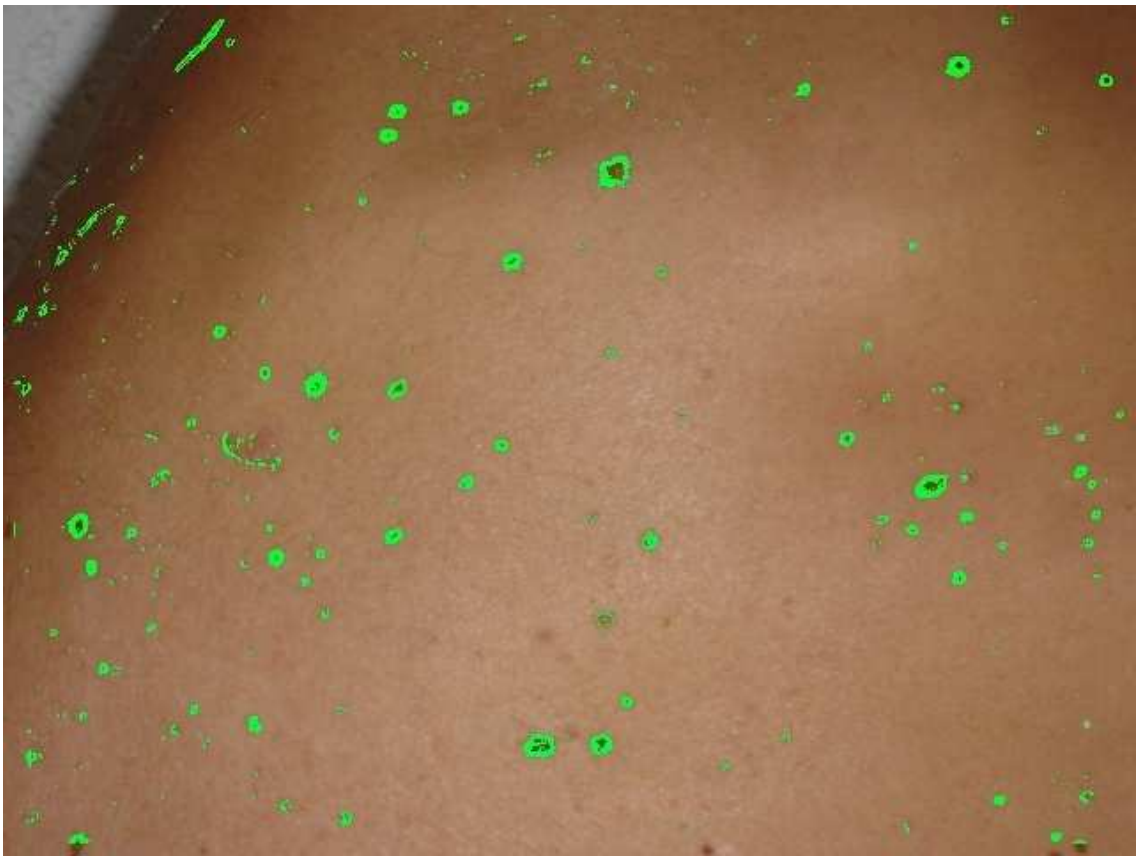


Fig. 9.3.4.e: Ejemplo antes de aplicar la extracción de contornos.

Fig. 9.3.4.f: Ejemplo tras aplicar la extracción de contornos.

Como se explicó en el apartado 3.3 que describía el algoritmo para el segundo procesado, aplicaremos esta función en tres ocasiones. En primer lugar sobre la imagen marcada sin aplicar ningún filtro antes de dar la primera solución definitiva. Después se volverá a utilizar antes de aplicar el filtro B. Y por último, se aplicará la función por tercera vez a la imagen que ha sido filtrada para eliminar el vello.

9.3.5-ESTUDIO DE CONTORNOS

Hasta ahora tenemos señalados los contornos de lo que consideramos lunares y les hemos aplicado la extracción de contornos. Aunque hemos desechado algunas marcas por su color o por su tamaño, todavía quedan marcas que son erróneas. Son ejemplos de marcas en torno a las axilas, vello, bordes debidos a la anatomía humana,… Para poder dar un resultado óptimo, se ha diseñado la siguiente función, que trata de eliminar ciertas marcas atendiendo a su forma. Por ejemplo nos permitirá desechar marcas en las axilas porque serán marcas muy grandes en comparación a los lunares, además de ser muy alargadas.

En primer lugar generará una imagen que hará de buffer, para después recorrer la imagen binaria generada en los pasos anteriores mediante dos bucles anidados. Cuando llegue a un píxel blanco llamará a la función “cerrar contorno” (que estudiaremos en el aparatado siguiente). La función recorrerá el contorno al que pertenece el píxel blanco y dependiendo de la forma de este contorno, incluirá o no el píxel blanco dentro del buffer. Cuando se termine de recorrer la imagen, se cargará la imagen guardada en el buffer. El esquema de funcionamiento será el siguiente:

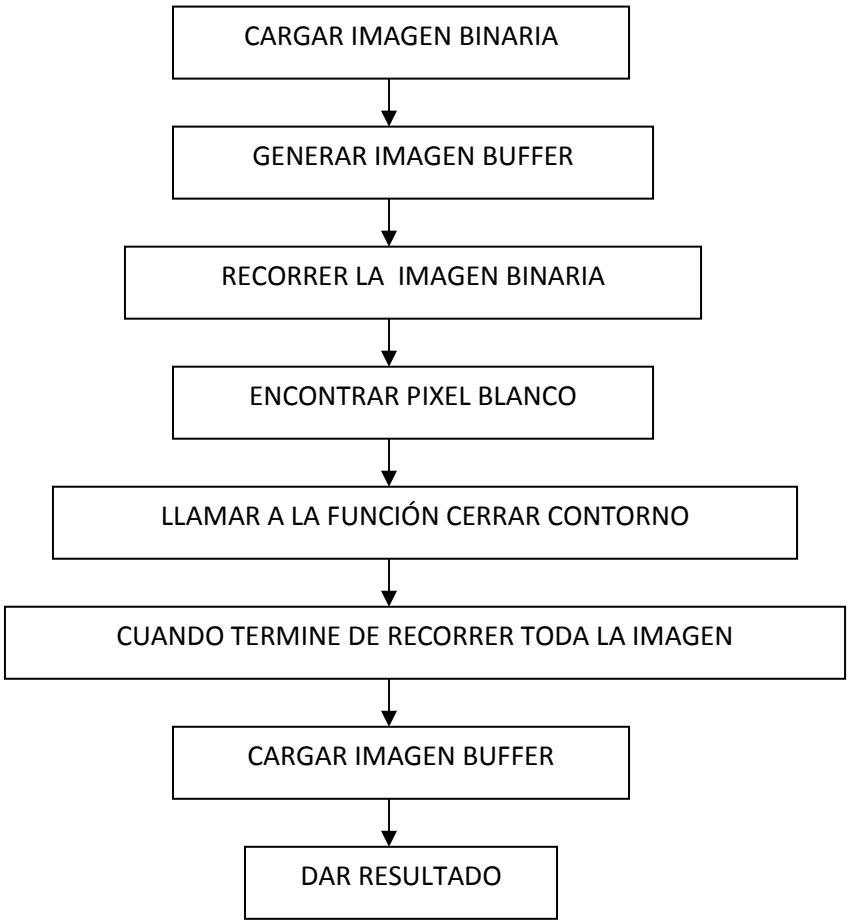


Fig. 9.3.5.a: Esquema de la función “Estudio de contornos”

El resultado será otra imagen binaria corregida, que se utilizará en el programa procesado2 para volver a marcar la imagen original. Para entender el funcionamiento exacto de esta función tendremos que estudiar la función a la que hace referencia ("cerrar contorno"). Se decidió hacerlo en dos funciones para poder mantener el buffer aunque se repita todo el procesado para cada píxel blanco encontrado. Además facilita su análisis y la implementación. En el apartado siguiente estudiaremos la segunda función, así como los criterios que se han tenido en cuenta para desechar una marca u otra.

9.3.6- CERRAR LOS CONTORNOS (FILTRO B)

La función anterior generaba un buffer donde guardar las coordenadas de los píxels considerados como marcas además de llamar a la función que vamos a estudiar a continuación cada vez con que se encuentre con un píxel blanco de la imagen binaria. Una vez encontrado el píxel blanco, la función recorrerá los píxels contiguos, lo que permitirá en un lunar recorrer todo su contorno. Esto nos dará información sobre la forma del lunar para poder desechar marcas que por su tamaño o comparación entre anchura y altura no pueden tratarse de lunares (lo llamaremos filtro B). A continuación podemos ver esquema de funcionamiento:

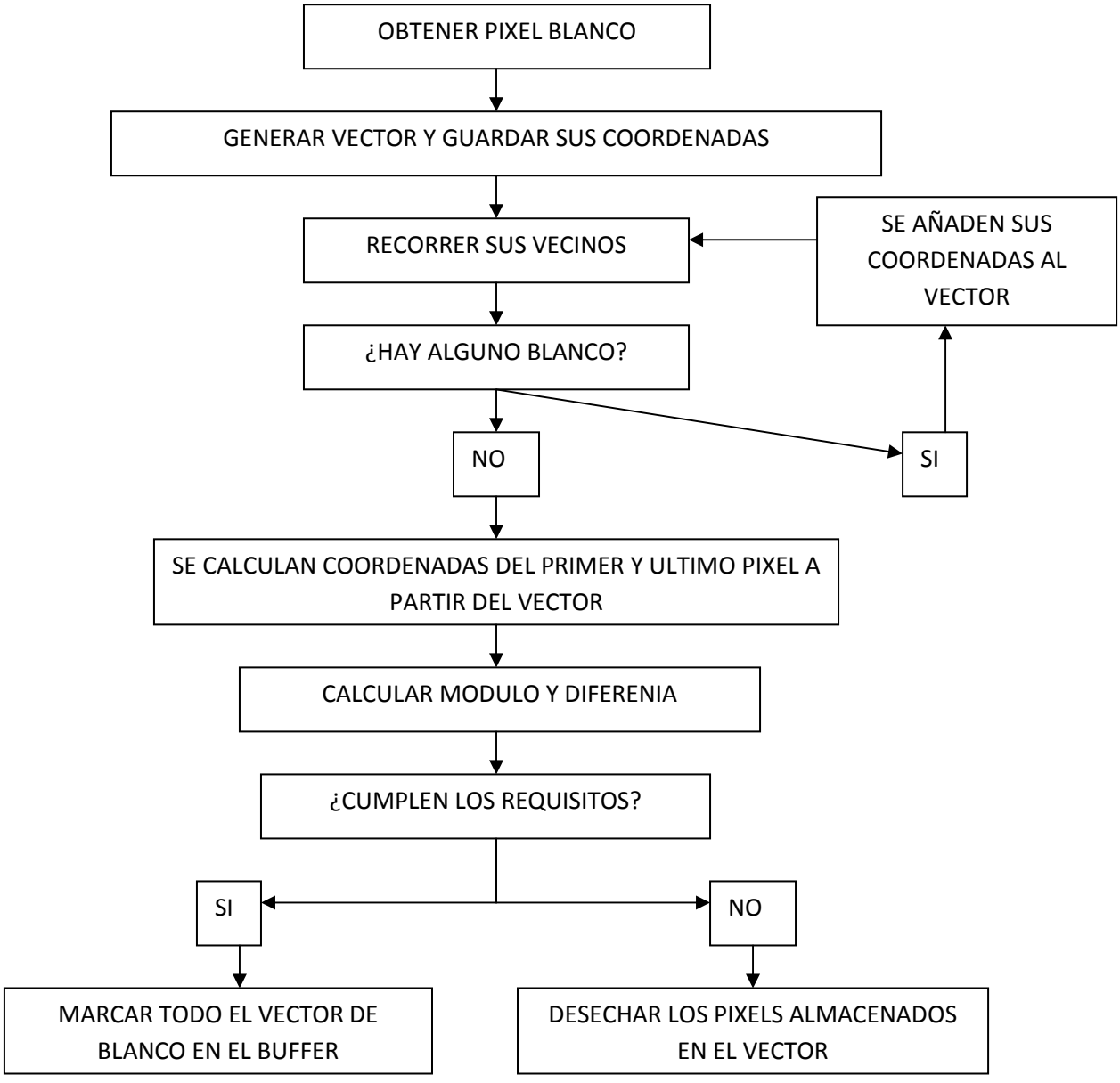


Fig. 9.3.6.a: Esquema de la función “Cerrar contornos”.

Se trata del algoritmo más complejo dentro del programa diseñado. Necesitará como entrada la imagen binaria y las coordenadas del píxel a analizar. Para empezar, el algoritmo guarda las coordenadas del píxel a estudiar en el inicio del vector. El siguiente paso será crear un bucle infinito que se seguirá ejecutando hasta que utilicemos dentro del bucle una condición que conlleve la expresión “break”.

Dentro de este bucle, comprobaremos si alguno de sus 8-vecinos es de color blanco. En caso afirmativo, añadiremos las coordenadas al vector. En caso contrario saldremos del bucle infinito para continuar con la función.

Como resultado tendremos un vector con las coordenadas correspondientes a cada píxel del contorno. Entonces se utilizarán dos características para desechar las marcas que no nos interesan, en primer lugar la diferencia entre la anchura y la altura del lunar y en segundo lugar la distancia entre el primer píxel analizado y el último. Analizaremos cómo utilizar estas dos propiedades para aplicarlas en nuestro procesado.

La primera característica es la diferencia entre anchura y altura de los lunares. Los lunares no suelen presentar formas muy alargadas al contrario de las marcas que podemos obtener de los bordes de una axila, hombros, etc. Las siguientes figuras lo ejemplifican claramente:

Fig. 9.3.6.b: Ejemplo de una marca producida por un lunar

Fig. 9.3.6.c: Ejemplo de una marca producida por el borde de una axila.

Podremos utilizar la propiedad de que la marca producida por la axila es mucho más ancha que alta, a diferencia del lunar, en la que no encontramos mucha diferencia. Realizando un estudio de la forma de los lunares obtuvimos los resultados de la tabla de la figura 9.3.6.d en cuanto a la diferencia ente los lados (medido en píxels):

Lunar nº	**Lado largo**	**Lado corto**	**Diferencia**
1	7	3	4
2	8	5	3
3	8	6	2
4	9	8	1
5	9	7	2
6	9	6	3
7	9	6	3
8	9	7	2
9	10	8	2
10	10	4	6
11	11	6	5
12	11	10	1
13	11	11	0
14	13	13	0
15	13	7	6
16	14	9	5
17	14	13	1
18	14	8	6
19	14	9	5
20	14	9	5
21	14	4	10
22	15	14	1
23	15	13	2
24	16	15	1

Lunar nº	Lado largo	Lado corto	Diferencia
25	16	16	0
26	17	13	4
27	18	13	5
28	20	19	1
29	33	32	1
30	34	32	2

Fig. 9.3.6.d: Tabla con los tamaños de los lunares.

Deducimos que la diferencia máxima entre el lado de mayor longitud respecto al de menor longitud no supera los 10 píxels. Al final se decidió por reducir esta diferencia a 5 píxels porque en la práctica dio mejores resultados.

Para obtener esta diferencia, calcularemos las coordenadas de los píxels más distantes verticalmente y las coordenadas para los píxels más distantes. Restando las respectivas coordenadas obtendremos la anchura y la altura del lunar. Y si restamos los valores obtendremos la diferencia.

La segunda cualidad será la distancia entre el primer y el último píxel analizado. Los lunares son representados por marcas que rodean su forma, generando un contorno cerrado. Ciertas marcas producidas por vello y otro tipo de bordes, generan contornos abiertos como se muestra en la siguiente imagen:

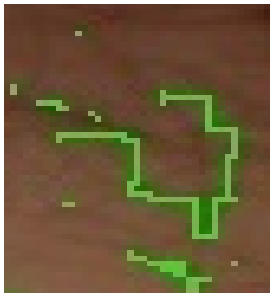


Fig. 9.3.6.e: Ejemplo de marca que no corresponde a un lunar y no se ha cerrado.

La propiedad anterior no sería capaz de desecharla porque la diferencia entre altura y anchura no va a ser mayor de 5 píxels. Sin embargo, hay mucha distancia ente el píxel inicial y el final. Utilizaremos el módulo para calcular la distancia entre estos dos píxels y si es mayor de lo estimado, la marca será desechada. Siendo (A1,B1) las coordenadas del primer píxel y (A2,B2) las del segundo, el módulo o longitud del vector formado por los dos puntos se calculará de la siguiente forma:

$$módulo = \sqrt{(A1 - A2)^2 + (B1 - B2)^2}$$

Empíricamente se obtuvieron resultados óptimos utilizando como referencia una distancia de 10 píxels. Si se obtienen distancias menores de 10 píxels se considerará un lunar, mientras que si la distancia es mayor, será considerada una marca errónea.

Para el cálculo del módulo, extraeremos del vector las coordenadas del primer y último píxel, aplicaremos la fórmula y tendremos la distancia. Pasaremos a comprobar si la marca cumple con las características ya mencionados. Guardaremos el resultado en la imagen que actúa como buffer y podremos volver a la función “Estudio del contorno” para comprobar la siguiente marca.

10-ESTUDIO DE LOS RESULTADOS

A continuación realizaremos un análisis de los resultados obtenidos aplicando el programa sobre diferentes imágenes. Para el desarrollo del programa hemos contado con 7 imágenes de diferentes partes del cuerpo, que utilizaremos también para el análisis de los resultados. Separaremos cada imagen en un apartado y analizaremos los problemas y los resultados obtenidos, para finalmente dar una idea global de la eficacia del programa. Para obtener los resultados, hemos aplicado el programa con sus 3 métodos, sobre cada una de las imágenes y hemos seleccionado la imagen con mejores resultados de las tres soluciones posibles. Sobre las imágenes seleccionadas hemos diferenciado 5 marcas posibles:

1º-Detecciones completas: las hemos marcado en verde y son las marcas que rodean perfectamente al lunar, dejando ver su interior para el estudio.

2º-Detecciones incompletas: marcadas en azul, representan lunares que han sido detectados pero no han sido perfectamente rodeados.

3º-Errores: De color amarillo, son marcas que no contienen lunares.

4º-Ruido: Píxels sueltos marcados erróneamente (casi siempre producidos por vello o por la textura de la piel). Los hemos marcado en color rojo. Deberemos establecer un criterio para diferenciar las marcas erróneas del ruido. Se decidió que consideraremos errores aquellas marcas que formen una forma cerrada cuyo interior no contenga un lunar válido, mientras que el ruido estará formado por líneas o puntos sueltos, que no llegan a formar una estructura cerrada. El ruido lo calcularemos por número de píxels, por ello para calcularlo, primero se pintó los píxels considerados ruido de color rojo y se diseñó un programa que recorriera las imágenes contando estos píxels rojos.

5º-Marcas que faltan: lunares que no han sido marcados. Las hemos señalado de color rosa.

Para facilitar el análisis calcularemos el porcentaje de detecciones completas, e incompletas, de errores y de marcas que faltan respecto a los lunares totales, y el porcentaje de ruido respecto a los píxels totales. Procederemos a continuación a analizar los resultados para la primera imagen.

10.1-RESULTADOS PARA LA IMAGEN 1

La primera imagen a procesar será la mostrada en la figura 10.1.a:

Fig. 10.1.a: Imagen 1

En ella podemos encontrar problemáticas de varios tipos: fondo de color grisáceo; bordes producidos por espalda, brazos etc.; vello, sombras, etc. Seleccionamos la imagen resultado de aplicar el filtro B, que marcó casi todos los lunares aunque dejó algún error y algún lunar por marcar. El resultado podemos verlo en la imagen 10.1.b:

Fig. 10.1.b: Mejor resultado para la imagen 1.

En la figura 10.1.c mostramos cómo hemos catalogado cada marca, hemos añadido las que faltaban y hemos puesto la leyenda junto con los resultados obtenidos:

Imagen 1	
Píxels Totales	307200
Marcas Totales	61
Detecciones Completas	48
% Detecciones Completas	78,69
Detecciones Incompletas	6
% Detecciones Incompletas	9,84
Errores	3
% Errores	4,92
Faltan	7
% Faltan	11,48
Ruido	1084
% Ruido	0,35

Fig.10.1.c: Imagen 1 con las marcas catalogadas y su respectiva leyenda con los resultados

Esta imagen presenta mucha problemática derivada de bordes, sombras y un poco de vello, por ello se ha escogido el resultado con el filtro B. Este filtro nos permite eliminar muchas marcas erróneas y ruido a costa de perder o detectar parcialmente alguna de las marcas más pequeñas. Esto se ve reflejado en los resultados, puesto que un 11,48% de los lunares no han sido detectados, y un 9,84 % han sido detectados parcialmente. Eso sí, gracias al filtro B, tan sólo hay 3 marcas erróneas y se ha reducido considerablemente el ruido producido por el vello, hombros, axilas, etc.

10.2-RESULTADOS PARA LA IMAGEN 2

La segunda imagen se puede observar en la figura 10.2.a:

Fig. 10.2.a: Imagen 2.

Podemos ver que el mayor obstáculo con el que nos encontramos es la cantidad de vello. Como en la imagen no aparecen bordes correspondientes a hombros o axilas, la imagen con mejor resultado es la correspondiente al filtro P. El resultado se puede ver en la figura 10.2.b:

Fig. 10.2.b: Resultado para la imagen 2.

Y en la figura 10.2.c encontramos las marcas catalogadas y su respectiva tabla:

Imagen 2	
Píxels Totales	307200
Marcas Totales	51
Detecciones Completas	41
% Detecciones Completas	80,39
Detecciones Incompletas	7
% Detecciones Incompletas	13,73
Errores	31
% Errores	60,78
Faltan	3
% Faltan	5,88
Ruido	1047
% Ruido	0,34

Fig.10.2.c: Imagen catalogada y leyenda con los resultados para la imagen 2.

En esta imagen nos hemos topado con la mayor dificultad que hemos encontrado a la hora de diseñar los algoritmos: la aparición de vello. El vello produce zonas de alta frecuencia que son detectadas por el filtrado Prewitt, además de que contiene colores semejantes a los de los lunares. Por ello en el segundo procesado utilizamos sobre todo sus características de forma para poder eliminarlo. El algoritmo consigue detectar casi el 95% de los lunares, aunque el 13,73% son detecciones incompletas debido al filtro P, que elimina parte de algunas marcas debido sobre todo a su proximidad a la zona de vello. A continuación mostramos en la figura 10.2.d un ejemplo de lunares que han sido detectados parcialmente porque el filtro P ha eliminado parte de la marca:

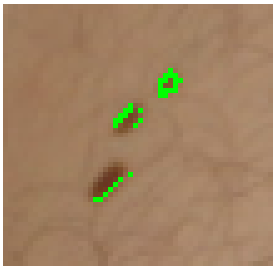


Fig. 10.2.d: Ejemplo de marcas parciales.

El filtro P ha actuado bastante bien, eliminando casi la totalidad del vello de la parte central de la imagen. De todas formas, encontramos un 60,71% de marcas erróneas producidas sobre todo por el vello que rodea las aréolas.

10.3-RESULTADOS PARA LA IMAGEN 3

Como podemos observar en la figura 10.3.a, la tercera imagen es muy similar a la segunda:

Fig. 10.3.a: Imagen 3.

En este caso el mayor problema vuelve a ser el vello presente en abundancia en la parte central de la imagen. Además aparecen bordes producidos por el brazo y el hombro en la parte derecha, complicando un poco más el procesado. Se trata de la imagen con más dificultades que nos hemos encontrado a priori. Los resultados obtenidos son mostrados en la figura 10.3.b:

.

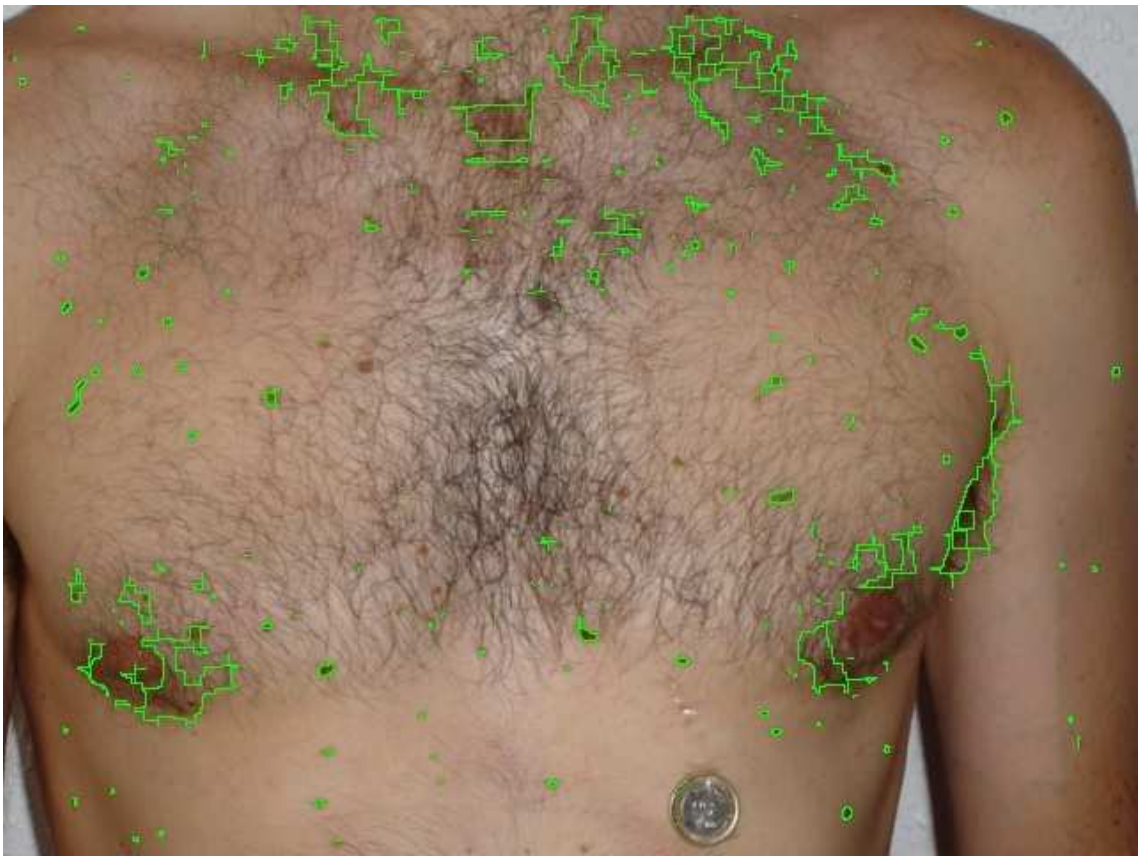


Fig. 10.3.b: Resultado obtenido al procesar la imagen 3.

A continuación presentamos el análisis de cada marca con su respectiva leyenda y sus datos en la figura 10.3.c:

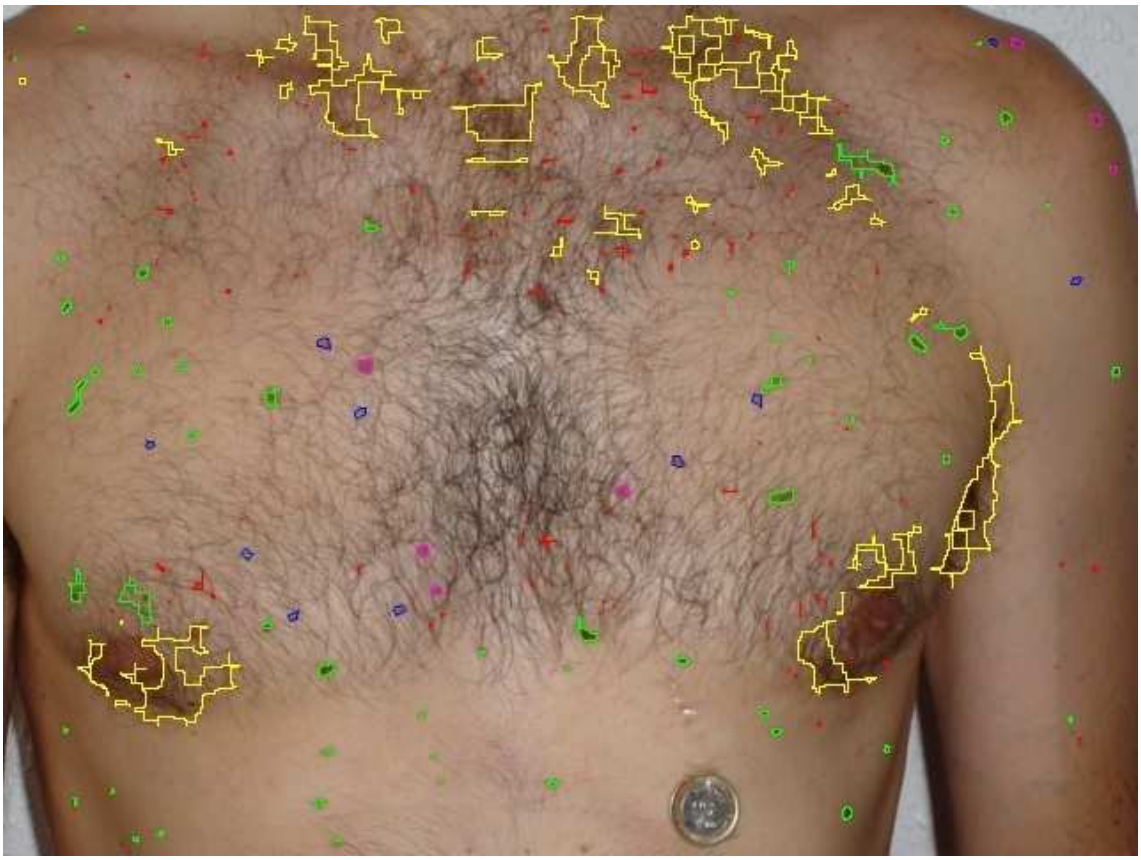


Imagen 3	
Píxels Totales	307200
Marcas Totales	73
Detecciones Completas	56
% Detecciones Completas	76,71
Detecciones Incompletas	10
% Detecciones Incompletas	13,70
Errores	38
% Errores	52,05
Faltan	7
% Faltan	9,59
Ruido	798
% Ruido	0,26

Fig. 10.3.c: Análisis de los resultados y leyenda de la imagen 3.

Los resultados son similares a los obtenidos para la imagen 2. El mejor resultado fue obtenido con el filtro P. Esto eliminó las marcas producidas por la abundancia de vello en la parte central de la imagen, a costa de obtener un 13,70% de marcas incompletas y un 9,59% que faltan. Aún así, se han detectado el 90,41% de los lunares. Como en la imagen 2, también hemos tenido bastantes marcas erróneas (52,05%), producidas también por el vello de las aréolas además de vello en la parte superior de la imagen y algún error en la axila de la parte derecha de la imagen. Estos resultados nos muestran que si una imagen contiene mucho vello y bordes, obtendremos mejor resultado con el filtro P.

10.4-RESULTADOS PARA LA IMAGEN 4

En la figura 10.4.a se muestra la cuarta imagen:

Fig. 10.4.a: Imagen 4.

Como se puede ver, la mayor dificultad de esta imagen radica en la aparición de bordes correspondientes a hombros, axilas y brazos. El resultado seleccionado es el obtenido para el filtro B. La imagen obtenida se puede observar en la figura 10.4.b:

Fig. 10.4.b: Resultado obtenido para la imagen 4.

En la figura 10.4.c mostramos el análisis de resultados, así como la leyenda correspondiente, incluyendo los datos obtenidos:

Imagen 4	
Píxels Totales	307200
Marcas Totales	164
Detecciones Completas	149
% Detecciones Completas	90,85
Detecciones Incompletas	13
% Detecciones Incompletas	7,93
Errores	7
% Errores	4,27
Faltan	2
% Faltan	1,22
Ruido	727
% Ruido	0,24

Fig. 10.4.c: Imagen con el análisis de resultados y su respectiva leyenda.

Para esta imagen hemos obtenido muy buenos resultados. Se han obtenido un 90,85% de los lunares completamente y un 7,93% parcialmente. El 1,33% restante, son dos pequeñas marcas que se han perdido por aplicar el filtro B. Sin embargo el filtro B nos ha permitido obtener tan sólo un 4,27% de errores, concentrados la mayoría en la zona de las axilas. Hemos sacrificado una parte insignificante de las marcas en los lunares por eliminar la mayoría de las marcas erróneas.

10.5-RESULTADOS PARA LA IMAGEN 5

Pasaremos a analizar los resultados para la quinta imagen, que se muestra en la figura 10.5.a:

Fig. 10.5.a: Imagen 5.

La imagen presenta pocas complicaciones. La mayor dificultad se puede encontrar en la moneda y en el borde de la parte inferior izquierda de la imagen. El resultado obtenido se puede ver en la figura 10.5.b:

Fig. 10.5.b: Resultados para la imagen 5.

A continuación mostraremos (en la figura 10.5.c) la imagen con el análisis de resultados y su leyenda con los datos numéricos obtenidos:

Imagen 5	
Píxels Totales	307200
Marcas Totales	97
Detecciones Completas	97
% Detecciones Completas	100,00
Detecciones Incompletas	0
% Detecciones Incompletas	0,00
Errores	6
% Errores	6,19
Faltan	0
% Faltan	0,00
Ruido	360
% Ruido	0,12

Fig. 10.5.c: Análisis de los resultados y tabla con la leyenda.

En esta imagen hemos obtenido unos resultados excelentes. El 100% de los lunares han sido detectados complemente gracias sobre todo a que no ha sido necesario utilizar los filtros B y P. Se ha cometido algún pequeño error debido a la moneda, a la axila y a la textura de la piel en la parte izquierda de la imagen. También ha contribuido a tener estos buenos resultados el hecho de que esta imagen ha sido tomada a una distancia menor que las imágenes anteriores. Esto permite obtener unos lunares de mayor tamaño y por tanto que sean más fáciles de detectar.

10.6-RESULTADOS PARA LA IMAGEN 6

En la figura 10.6.a mostramos la sexta imagen:

Fig.10.6.a: Imagen 6.

Se trata de una imagen muy sencilla, en la que no encontramos vello y que el único borde que puede dar problemas se sitúa en la parte superior izquierda de la imagen. El resultado obtenido se puede ver en la figura 10.6.b:

Fig.10.6.b: Resultado de procesar la imagen 6.

La imagen con el análisis de resultados y su correspondiente leyenda con los datos obtenidos son mostrados en la figura 10.6.c:

Imagen 6	
Píxels Totales	307200
Marcas Totales	86
Detecciones Completas	86
% Detecciones Completas	100,00
Detecciones Incompletas	0
% Detecciones Incompletas	0,00
Errores	0
% Errores	0,00
Faltan	0
% Faltan	0,00
Ruido	399
% Ruido	0,13

Fig. 10.6.c: Análisis de resultados y tabla de la imagen 6.

En esta imagen hemos obtenido unos resultados óptimos gracias a que no aparece vello ni bordes importantes. Se han marcado completamente el 100% de los lunares gracias a que no ha hecho falta utilizar ninguno de los filtros. Pese a no utilizar los filtros, no hemos obtenido ningún error, ni siquiera en el borde de la parte superior izquierda de la imagen. Al igual que en la imagen 5, la imagen ha sido tomada a menor distancia, obteniendo mejores resultados que en imágenes realizadas a mayor distancia.

10.7-RESULTADOS PARA LA IMAGEN 7

A continuación mostramos en la figura 10.7.a la séptima y última imagen utilizada para el análisis de los resultados:

Fig. 10.7.a: Imagen 7.

Se trata de una imagen similar a la imagen 5. No encontramos problema con el vello ni con los bordes, pero la moneda puede darnos alguna dificultad. Los resultados obtenidos los mostramos en la figura 10.7.b:

Fig. 10.7.b: Resultados obtenidos para la imagen 7.

A continuación mostraremos en la figura 10.7.c el análisis de resultados con su correspondiente leyenda incluyendo los datos obtenidos:

Imagen 7	
Píxels Totales	307200
Marcas Totales	97
Detecciones Completas	97
% Detecciones Completas	100,00
Detecciones Incompletas	0
% Detecciones Incompletas	0,00
Errores	4
% Errores	4,12
Faltan	0
% Faltan	0,00
Ruido	233
% Ruido	0,08

Fig.10.7.c: Análisis de los resultados y leyenda para la imagen 7.

Como era de esperar, los resultados han sido similares a los obtenidos en la quinta imagen. Al no necesitar de los filtros de B y P, se han marcado totalmente el 100% de los lunares. Se han producido 4 errores, 2 que se deben a la moneda y otros dos que se deben a pequeñas sombras de la piel. El gran tamaño de los lunares también ha contribuido a estos buenos resultados.

10.8-RESULTADOS GLOBALES

Por último, hemos recogido los datos obtenidos del análisis de las siete imágenes en una tabla y hemos calculado medias y desviaciones. La tabla obtenida se puede observar a continuación, en la figura 10.8.a:

IMAGEN	PIXELS TOTALES	LUNARES	DETECCIONES COMPELTAS		DETECCIONES INCOMPLETAS		LUNARES DETECTADOS	
			NUMERO	% RESPECTO LUNARES	DETECCIONES	% RESPECTO LUNARES	DETECCIONES	% RESPECTO LUNARES
1	307200	61	48	78,69	6	12,5	54	88,52
2	307200	51	41	80,39	7	17,07	48	94,12
3	307200	73	56	76,71	10	17,86	66	90,41
4	307200	164	149	90,85	13	8,72	162	98,78
5	307200	97	97	100	0	0,00	97	100,00
6	307200	86	86	100,00	0	0,00	86	100,00
7	307200	97	97	100,00	0	0,00	97	100,00
Media	307200	89,86	82	91,25	5,14	6,27	87,14	96,98
Desviación	0	37,06	37,57	10,77	5,30	8,09	38,46	4,94

IMAGEN	PIXELS TOTALES	LUNARES	ERRORES		RUIDO		LUNARES QUE FALTAN	
			ERRORES	% RESPECTO A LUNARES	PIXELS	% RESPECTO A PIXELS TOTALES	CANTIDAD	% RESPECTO A LUNARES
1	307200	61	3	4,92	1084	0,35	7	11,48
2	307200	51	31	60,78	1047	0,34	3	5,88
3	307200	73	38	52,05	798	0,26	7	9,59
4	307200	164	7	4,27	727	0,24	2	1,22
5	307200	97	6	6,19	360	0,12	0	0,00
6	307200	86	0	0,00	399	0,13	0	0,00
7	307200	97	4	4,12	233	0,08	0	0,00
Media	307200	89,86	12,71	14,15	664	0,22	2,71	3,02
Desviación	0	37,06	15,18	25,82	339,95	0,11	3,15	4,94

Fig. 10.8.a: Tabla con los resultados obtenidos para las 7 imágenes procesadas.

Podemos ver que de media, más del 96% de los lunares han sido detectados. La cifra oscila entre el 88,52% en la imagen 1, hasta el 100% que hemos encontrado en varias imágenes. La problemática de la imagen 1 es la aparición de muchos bordes, por ello se tuvo que utilizar el filtro P, perdiendo el 11,48% de los lunares. Sin embargo, en las imágenes 5, 6 y 7 hemos obtenido unos resultados del 100% debido a que los lunares son de mayor tamaño además de que no tenemos problemas de bordes, vello, sombras, etc.

Otro dato significativo es el número de detecciones incompletas. La media nos indica que el 6,27% de los lunares han sido marcados parcialmente. Este porcentaje oscila entre el 0% en las imágenes 5,6 y 7 (ya que el 100% son marcas completas) y el 17,86% de la imagen 3. Dado que la imagen 2 nos ha dado un 17,07% concluimos que el filtro P tiene el defecto de eliminar parcialmente algunas de las marcas (recordemos que las imágenes 2 y 3 son las que más vello presentaban). Los mejores resultados volvemos a encontrarlos en las imágenes con los lunares de mayor tamaño y con menor problemática.

Pasaremos a estudiar el número de marcas erróneas presentes en los resultados. De nuevo encontramos los peores resultados en las imágenes 2 y 3, en las que encontramos un 60,78% y un 52,05% respectivamente. Aunque el filtro P haya conseguido eliminar la gran mayoría de marcas erróneas, todavía quedan bastantes marcas que no podemos eliminar ajustando el filtro P porque perdemos demasiadas marcas correctas. De todas formas, son marcas de gran tamaño que podrían ser fácilmente eliminadas en un postprocesado posterior. El resto de fotografías presentan en torno al 5% de errores, que corresponden a pequeñas marcas producidas por bordes y textura de la piel.

Para finalizar este estudio de resultados, nos centraremos en el número de píxels ruidosos. Si lo comparamos con el número total de píxels de las imágenes, nos encontramos con una media del 0,22% con una desviación del 0,11%. Se tratan de píxels sueltos que quedan marcados debido a bordes y textura de la piel, o de marcas erróneas que los filtros P y B no han conseguido eliminar totalmente. No suponen un gran problema, puesto que en un postprocesado serían fáciles de eliminar por tratarse de píxels sueltos.

11-TIEMPOS DE CÓMPUTO

El programa utilizará precisa de muchos cálculos y por ello en equipos antiguos puede demorarse demasiado, llegando incluso a bloquearlo. Por ello se hizo necesario dedicar un apartado al estudio de los tiempos de cómputo en relación a la potencia del PC utilizado. Para realizar las pruebas, utilizamos un sistema operativo Windows XP, con la versión de Matlab 6.5.0.180913a Release 13. Cronometramos el tiempo que necesita el programa para procesar las 7 imágenes que utilizamos para el análisis de los resultados, teniendo en cuenta que son imágenes en formato BMP con una resolución de 640x480 píxels (aproximadamente ocupan 1MB de espacio).

Primero hicimos las pruebas con un PC Pentium III a 800Mhz y con una memoria RAM de 128 MB. Los tiempos de cómputo alcanzaron los 10 minutos y por ello se decidió que eran los requisitos mínimos para ejecutar el programa, puesto que en un PC inferior el tiempo de ejecución sería excesivo. Además, teniendo abiertos otros programas el PC llegó a bloquearse, por eso se recomienda no tener abiertos otros programas mientras se realiza el procesado y desechamos totalmente la idea de utilizar PCs de menor potencia.

El siguiente equipo utilizado es un Pentium IV a 2,4 GHz con 256MB de RAM. Los tiempos de cómputo se redujeron hasta los 6 minutos, además de que teniendo abiertos otros programas no se produjeron bloqueos. El tiempo se ha reducido muchísimo, pero se tornará excesivo si queremos procesar muchas imágenes.

Para finalizar probamos con dos equipos con procesador de doble núcleo (Pentium D y Pentium Core 2 Duo) y 1GB de RAM. Los tiempos se redujeron hasta los 2 minutos, por ello recomendamos utilizar equipos de estas características o superiores para utilizar el programa, sobre todo si se va a trabajar con muchas imágenes.

12- CONCLUSIONES

En el siguiente apartado mostraremos las conclusiones a las que hemos llegado tras el diseño y el estudio del programa para la detección de lunares. Después estudiaremos si los resultados eran los esperados y la utilidad del programa. Las conclusiones obtenidas fueron las siguientes:

1º- Al utilizar un filtrado de bordes para la detección, las zonas con alta frecuencia que no corresponden a lunares, nos dieron problemas. Para evitarlos tuvimos que realizar un segundo procesado eliminando marcas erróneas.

2º- Los bordes correspondientes a axilas, brazos, hombros,… pueden producir marcas erróneas aunque las hemos eliminado la mayoría a costa de perder marcas pequeñas. Se consigue reducir los errores a un 5% a cambio de perder un 11% de los lunares.

3º- El mayor problema encontrado ha sido el vello. Si se presenta en abundancia, el mejor resultado será el obtenido con el filtro P. Aún así pueden aparecer todavía marcas erróneas producidas por vello (cerca del 35%) y se pueden perder algunas de las marcas correspondientes a lunares o quedar marcas completas como marcas parciales.

4º- Cuando en la fotografía a procesar encontramos mucho vello y bordes producidos por brazos, cuello,… es preferible utilizar el filtro P que el filtro B. El compromiso entre marcas erróneas eliminadas y marcas buenas perdidas es mucho mejor.

5º- Tras el procesado, aparecen pequeños píxels marcados correspondiente a ruido. Se trata de algo insignificante (en torno al 0,22%) y que en un procesado posterior sería fácil de eliminar.

6º- Para imágenes sin vello ni bordes que no correspondan a lunares, tendremos resultados óptimos. No será necesario utilizar ninguno de los filtros diseñados y por tanto se marcarán totalmente el 100% de las pecas.

7º-Cuanto mayor es el tamaño de los lunares, mejor resultado obtenemos. Por ello es aconsejable realizar la fotografía desde cerca o aplicando un zoom sobre los lunares.

Teniendo en cuenta todo esto podemos decir que los resultados son los esperados. La detección media es de un 97%, bajando como mucho hasta el 88% en imágenes con mucha problemática en cuanto a bordes. En imágenes con vello se detecta más del 90% aunque se cometen errores. Para que el programa sea útil a nuestros objetivos iniciales, será necesario un postprocesado para eliminar marcas erróneas y ruido, así como implementar un algoritmo para que enumere las marcas y que las compare con estudios anteriores para ver su evolución. Estos algoritmos no pudieron ser implementados por falta de tiempo y por la complejidad de los algoritmos ya expuestos. Todo esto lo expondremos en el capitulo siguiente dedicado a dar sugerencias para líneas futuras.

13- LINEAS FUTURAS

Como explicamos en el capítulo anterior, el programa diseñado no será capaz de enumerar las marcas y compararlas con marcas de una peca. Por falta de tiempo y recursos nos quedamos en la detección de las marcas. Para continuar el trabajo realizado, daremos las siguientes sugerencias:

1º- Diseñar un post procesado que permita eliminar marcas manualmente. Esto permitirá eliminar todos los errores producidos por el vello y otras zonas de alta frecuencia. También nos permitirá eliminar el ruido generado o incluso marcas correctas pero que sean insignificantes por su tamaño.

2º- Diseño de un algoritmo que enumere las marcas. Deberá ser capaz de contar el número de marcas en la imagen y de asignar un número a cada una. Así luego se podrán segmentar y estudiar cada marca por separado.

3º- Diseño de un programa que compare las fotografías segmentadas en el programa anterior. Debe ser capaz de comparar dos imágenes similares de una misma parte del cuerpo, diciendo si han aparecido nuevas marcas, si ha habido cambios en las anteriores, etc.

Para facilitar el proceso, también se podría sacar una foto por cada lunar. Al ser fotos de lunares de gran tamaño el algoritmo diseñado los detectaría sin problemas, pudiendo prescindir de las dos primeras sugerencias.

También se podría mejorar la detección del programa ya diseñado. Para ello sugerimos trabajar con más fotos, que tengan mejor resolución y que los lunares tengan el mayor tamaño posible. Además se podría crear una interfaz que permitiera ajustar la sensibilidad de los filtros B y P.

14-BIBLIOGRAFÍA Y REFERENCIAS

A hora de desarrollar el proyecto, ha sido necesaria bibliografía sobre la utilización de Matlab y programación con esta aplicación, bibliografía sobre procesamiento digital de imagen en general y por último sobre la utilización de Photoshop:

-APRENDA MATLAB 5.2 COMO SI ESTUVIERA EN PRIMERO. Javier García de Jalón, José Ignacio Rodríguez, Alfonso Barzález, Patxi Funes, Alberto Larrazábal. Universidad de Navarra, Escuela Superior de Ingenieros Industriales. San Sebastián 1998

-APRENDA MATLAB 6.5 COMO SI ESTUVIERA EN PRIMERO. Javier García de Jalón. Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros Industriales. Madrid 2004.

-MANUAL IMPRESCINDIBLE ADOBE PHOTOSHOP 7. José María Delgado Cabrera. Ed. Anaya Multimedia 2002.

-TRATAMIENTO DIGITAL DE IMÁGENES. Rafael C. González, Richard E. Woods. Ed. Addison 1996.

Para más información se acerca de otros programas similares, se puede acceder a las siguientes direcciones:

http://plutarco.disca.upv.es/~jcperez/doctorado/SV2D3DPI/trabajos/Medicina/melanomas.html

http://www.molemap.co.nz

ANEXO1: IMPLEMENTACION DE LOS PROGRAMAS

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%                   FUNCION INTERFAZ
%   Muestra por pantalla el cuadro de dialogo donde introducir la
%   imagen. Esperara a la pulsación de intro para mostrar la
%   imagen original llamar a las dos partes del procesado y mostrar
%   los resultados. En el caso de que haya algún tipo de error mostrará
%   una pantalla indicándolo.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = interfaz(varargin)

%Parte de la función generada automáticamente por Matlab
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @interfaz_OpeningFcn, ...
                   'gui_OutputFcn',  @interfaz_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function interfaz_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
```

```
guidata(hObject, handles);
function varargout = interfaz_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function edit1_Callback(hObject, eventdata, handles)



%Cuando se introduzca la imagen y se pulse intro se ejecutara el siguiente código


try %Si no hay errores se ejecutaran las funciones comprendidas entre try y catch
a=get(hObject,'String'); %Se leen los datos introducidos en el cuadro de dialogo
foto=imread(a);
imwrite(foto,'original.bmp'); %Se carga la imagen original y se realiza una copia
procesando %se muestra la imagen original
procesado1(foto); %El programa llama a las dos partes del procesado
Proceso1=imread('Procesado1.bmp');
procesado2(Proceso1,foto);
alert=wavread('Alerta.wav'); %al terminar lanza una señal acústica
sound(alert);
Exito %Se muestra el dialogo de éxito y se cierran los demás
close('Detector')
close('Mensaje')
catch %En el caso de que algo falle se mostrara el mensaje de error y se cerraran los demás
diálogos
Error_en_imagen
close('Detector')
close('Mensaje')
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%                  FUNCION PROCESADO1
%   Función que realiza la primera parte del procesado, es decir aplicara
%   los filtros prewitt, realizara la umbralización y lo marcara sobre la
%   imagen original, guardando el resultado como 'procesado1.bmp'.Como
%   variable de entrada necesita la imagen original y como salida dará la
%   imagen marcada tras el primer procesado.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=procesado1(original)

procesando %muestra la imagen original mientras se realiza todo el proceso

f=original;
T=blanconegro(f);%lee la imagen original y la transforma a escala de grises

Ph=prewitt1(T); %aplica sobre la imagen los 4 filtros prewitt para las 4 direcciones
Pv=prewitt2(T);
Pd1=prewitt3(T);
Pd2=prewitt4(T);

umbra1=umbralizar(Ph); % umbraliza cada uno de los filtros por separado
umbra2=umbralizar(Pv);
umbra3=umbralizar(Pd1);
umbra4=umbralizar(Pd2);

 y=marca(f,umbra1); %marca la imagen con cada una de los filtros umbralizados
 y=marca(y,umbra2);
 y=marca(y,umbra3);
 y=marca(y,umbra4);

 imwrite(y,'Procesado1.bmp');%guarda los resultados
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        FUNCION BLANCONEGRO
%   Permite pasar de una imagen a color en el dominio RGB a una imagen en
%   escala de grises. Esto permitirá aplicar sobre la imagen procesados que
%   en RGB no se puede, por ejemplo un filtrado Prewitt. Como entrada se
%   necesita la imagen original y como salida dará la imagen en escala de
%   grises y la guardara como 'blanconegro.bmp'
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function T=blanconegro(f)

f=double(f); %transforma la imagen a double para poder realizar operaciones matemáticas

[M,N]=size(f);
N=N/3; %calcula las filas y las columnas de la imagen

for i=1:M
   for j=1:N %bucle anidado que recorre cada uno de los píxels que componen la imagen

      R=f(i,j,1);
      G=f(i,j,2);
      B=f(i,j,3); %obtiene el valor de las componentes R, G y B para cada pixel.

      T(i,j)=0.3*R+0.59*G+0.11*B;
      T(i,j)=round(T(i,j)); %calcula y redondea el valor de gris para cada pixel.
   end
end

T=uint8(T);
imwrite(T,'blanconegro.bmp'); %transforma la imagen a Uint8 y la guarda
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%                    FUNCION PREWITT1
%  Realiza un filtrado Prewitt horizontal sobre la imagen en escala
%  de grises para obtener otra imagen con los píxels correspondientes
%  a los bordes horizontales remarcados. Como entrada necesita la imagen en
%  escala de grises y como salida dará otra imagen en escala de grises
%  lista para umbralizar y marcar sobre la original
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function T=prewitt1(f)

f=double(f); % transforma imagen a double para poder realizar operaciones matemáticas
[M,N]=size(f); % calcula las dimensiones de la imagen

for i=1:M % doble bucle que va recorriendo cada pixel de la imagen
   for j=1:N

      % Se irá comprobando si los píxels corresponden a bordes u esquinas
      % de la imagen para ver cuántos vecinos habrá que tener en cuenta.
      % Después se multiplicara cada vecino por su coeficiente
      % correspondiente y se realizara la suma, obteniendo el valor del
      % filtro para ese pixel en concreto

      if i==1
         if j==1 % comprueba si el pixel pertenece a la esquina superior izquierda
            T(i,j)=f(i+1,j+1)+f(i+1,j);
         elseif j==N % comprueba si el pixel pertenece a la esquina superior derecha
            T(i,j)=f(i+1,j)+f(i+1,j-1);
         else % comprueba si el pixel pertenece a la pertenece a fila de arriba
            T(i,j)=f(i+1,j+1)+f(i+1,j)+f(i+1,j-1);
         end
```

```
        elseif i==M
            if j==1 % comprueba si el pixel pertenece a la esquina inferior izquierda
                T(i,j)=-f(i-1,j)-f(i-1,j+1);
            elseif j==N % comprueba si el pixel pertenece a la esquina inferior derecha
                T(i,j)=-f(i-1,j-1)-f(i-1,j);
            else % comprueba si el pixel pertenece a la pertenece a fila de abajo
                T(i,j)=-f(i-1,j-1)-f(i-1,j)-f(i-1,j+1);
            end
        else
            if j==1 % comprueba si el pixel pertenece a la  columna de la izquierda
                T(i,j)=-f(i-1,j)-f(i-1,j+1)+f(i+1,j+1)+f(i+1,j);
            elseif j==N % comprueba si el pixel pertenece a la  columna de la derecha
                T(i,j)=-f(i-1,j-1)-f(i-1,j)+f(i+1,j)+f(i+1,j-1);
            else % El pixel pertenece a cualquier otra parte de la imagen
                T(i,j)=-f(i-1,j-1)-f(i-1,j)-f(i-1,j+1)+f(i+1,j+1)+f(i+1,j)+f(i+1,j-1);
            end
        end
    end
end


T=escala256(T); % Tras los cálculos el rengo dinámico ha cambiado, se vuelve a meter en el
rango 0-256

T=uint8(T); % se transforma a uint8 para dar el resultado final
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%                        FUNCION PREWITT2
%  Realiza un filtrado Prewitt vertical sobre la imagen en escala
%  de grises para obtener otra imagen con los píxels correspondientes
%  a los bordes verticales remarcados. Como entrada necesita la imagen en
%  escala de grises y como salida dará otra imagen en escala de grises
%  lista para umbralizar y marcar sobre la original
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function T=prewitt2(f)

f=double(f); % transforma imagen a double para poder realizar operaciones matemáticas
[M,N]=size(f); % calcula las dimensiones de la imagen

for i=1:M % doble bucle que va recorriendo cada pixel de la imagen
   for j=1:N

        % Se irá comprobando si los píxels corresponden a bordes u esquinas
       % de la imagen para ver cuántos vecinos habrá que tener en cuenta.
       % Después se multiplicara cada vecino por su coeficiente
       % correspondiente y se realizara la suma, obteniendo el valor del
       % filtro para ese pixel en concreto

       if i==1
          if j==1 %comprueba si el pixel pertenece a la esquina superior izquierda
             T(i,j)=f(i,j+1)+f(i+1,j+1);
          elseif j==N %comprueba si el pixel pertenece a la esquina superior derecha
             T(i,j)=-f(i,j-1)-f(i+1,j-1);
          else %comprueba si el pixel pertenece a la  fila de arriba
             T(i,j)=-f(i,j-1)-f(i+1,j-1)+f(i,j+1)+f(i+1,j+1);
          end
```

```
        elseif i==M
            if j==1 %comprueba si el pixel pertenece a la esquina inferior izquierda
                T(i,j)=f(i-1,j+1)+f(i,j+1);
            elseif j==N %comprueba si el pixel pertenece a la esquina inferior derecha
                T(i,j)=-f(i-1,j-1)-f(i,j-1);
            else %comprueba si el pixel pertenece a la  fila de abajo
                T(i,j)=-f(i-1,j-1)-f(i,j-1)+f(i-1,j+1)+f(i,j+1);
            end
        else
            if j==1 %comprueba si el pixel pertenece a la columna de la izquierda
                T(i,j)=f(i-1,j+1)+f(i,j+1)+f(i+1,j+1);
            elseif j==N %comprueba si el pixel pertenece a la  columna de la derecha
                T(i,j)=-f(i-1,j-1)-f(i,j-1)-f(i+1,j-1);
            else %pertenece a cualquier otra parte de la imagen
                T(i,j)=-f(i-1,j-1)-f(i,j-1)-f(i+1,j-1)+f(i-1,j+1)+f(i,j+1)+f(i+1,j+1);
            end
        end
    end
end


T=escala256(T); % Tras los cálculos el rengo dinámico ha cambiado, se vuelve a meter en el
rango 0-256

T=uint8(T); % se transforma a uint8 para dar el resultado final
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                       FUNCION PREWITT3
%  Realiza un filtrado Prewitt de 45º sobre la imagen en escala
%  de grises para obtener otra imagen con los píxels correspondientes
%  a los bordes con 45º remarcados. Como entrada necesita la imagen en
%  escala de grises y como salida dará otra imagen en escala de grises
%  lista para umbralizar y marcar sobre la original
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function T=prewitt3(f)

f=double(f); % transforma imagen a double para poder realizar operaciones matemáticas
[M,N]=size(f); % calcula las dimensiones de la imagen

for i=1:M % doble bucle que va recorriendo cada pixel de la imagen
   for j=1:N

       % Se irá comprobando si los píxels corresponden a bordes u esquinas
       % de la imagen para ver cuántos vecinos habrá que tener en cuenta.
       % Después se multiplicara cada vecino por su coeficiente
       % correspondiente y se realizara la suma, obteniendo el valor del
       % filtro para ese pixel en concreto

       if i==1
          if j==1 %comprueba si el pixel pertenece a la esquina superior izquierda
             T(i,j)=f(i+1,j+1)+f(i+1,j)+f(i,j+1);
          elseif j==N %comprueba si el pixel pertenece a la esquina superior derecha
             T(i,j)=-f(i,j-1)+f(i+1,j);
          else %comprueba si el pixel pertenece a la pertenece a fila de arriba
             T(i,j)=-f(i,j-1)+f(i+1,j+1)+f(i+1,j)+f(i,j+1);
          end
```

```
        elseif i==M
            if j==1 %comprueba si el pixel pertenece a la esquina inferior izquierda
                T(i,j)=-f(i-1,j)+f(i,j+1);
            elseif j==N %comprueba si el pixel pertenece a la esquina inferior derecha
                T(i,j)=-f(i-1,j-1)-f(i-1,j)-f(i,j-1);
            else %comprueba si el pixel pertenece a la  fila de abajo
                T(i,j)=-f(i-1,j-1)-f(i-1,j)-f(i,j-1)+f(i,j+1);
            end
        else
            if j==1 %comprueba si el pixel pertenece a la  columna de la izquierda
                T(i,j)=-f(i-1,j)+f(i+1,j+1)+f(i+1,j)+f(i,j+1);
            elseif j==N %comprueba si el pixel pertenece a la columna de la derecha
                T(i,j)=-f(i-1,j-1)-f(i-1,j)-f(i,j-1)+f(i+1,j);
            else %pertenece a cualquier otra parte de la imagen
                T(i,j)=-f(i-1,j-1)-f(i-1,j)-f(i,j-1)+f(i+1,j+1)+f(i+1,j)+f(i,j+1);
            end
        end
    end
end


T=escala256(T); % Tras los cálculos el rango dinámico ha cambiado, se vuelve a meter en el
rango 0-256

T=uint8(T); % se transforma a uint8 para dar el resultado final
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        FUNCION PREWITT4
%  Realiza un filtrado Prewitt a 135º sobre la imagen en escala
%  de grises para obtener otra imagen con los píxels correspondientes
%  a los bordes con 135º remarcados. Como entrada necesita la imagen en
%  escala de grises y como salida dará otra imagen en escala de grises
%  lista para umbralizar y marcar sobre la original
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function T=prewitt4(f)

f=double(f); % transforma imagen a double para poder realizar operaciones matemáticas
[M,N]=size(f); % calcula las dimensiones de la imagen

for i=1:M % doble bucle que va recorriendo cada pixel de la imagen
   for j=1:N

       % Se irá comprobando si los píxels corresponden a bordes u esquinas
       % de la imagen para ver cuántos vecinos habrá que tener en cuenta.
       % Después se multiplicara cada vecino por su coeficiente
       % correspondiente y se realizara la suma, obteniendo el valor del
       % filtro para ese pixel en concreto

       if i==1
          if j==1 %comprueba si el pixel pertenece a la esquina superior izquierda
             T(i,j)=-f(i+1,j)+f(i,j+1);
          elseif j==N %comprueba si el pixel pertenece a la esquina superior derecha
             T(i,j)=-f(i+1,j-1)-f(i+1,j)-f(i,j-1);
          else %comprueba si el pixel pertenece a la pertenece a fila de arriba
             T(i,j)=-f(i+1,j-1)-f(i+1,j)-f(i,j-1)+f(i,j+1);
          end
       elseif i==M
```

```
        if j==1 %comprueba si el pixel pertenece a la esquina inferior izquierda
          T(i,j)=+f(i-1,j+1)+f(i-1,j)+f(i,j+1);
        elseif j==N %comprueba si el pixel pertenece a la esquina inferior derecha
          T(i,j)=-f(i,j-1)+f(i-1,j);
        else %comprueba si el pixel pertenece a la pertenece a fila de abajo
           T(i,j)=-f(i,j-1)+f(i-1,j+1)+f(i-1,j)+f(i,j+1);
        end
      else
        if j==1 %comprueba si el pixel pertenece a la  columna de la izquierda
          T(i,j)=-f(i+1,j)+f(i-1,j+1)+f(i-1,j)+f(i,j+1);
        elseif j==N %comprueba si el pixel pertenece a la  columna de la derecha
           T(i,j)=-f(i+1,j-1)-f(i+1,j)-f(i,j-1)+f(i-1,j);
        else %pertenece a cualquier otra parte de la imagen
           T(i,j)=-f(i+1,j-1)-f(i+1,j)-f(i,j-1)+f(i-1,j+1)+f(i-1,j)+f(i,j+1);
        end
      end
    end
end


T=escala256(T); % Tras los cálculos el rengo dinámico ha cambiado, se vuelve a meter en el
rango 0-256

T=uint8(T); % se transforma a uint8 para dar el resultado final
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       FUNCION ESCALA256
% Permite cambiar de cualquier rango dinámico al rango dinámico de la
% escala de gris (de 0 a 255). Como entrada necesita una matriz de valores,
% de los cuales calculara el rango y lo pasara al rango 0-256
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function T=escala256(f)

[M,N]=size(f); % Calcula las dimensiones de la matriz

maximo=max(f); % Calcula el máximo por filas
maximo=max(maximo); % Calcula el máximo global

minimo=min(f); % Calcula el mínimo por filas
minimo=min(minimo); % Calcula el mínimo global

for i=1:M % bucle anidado que recorre cada pixel de la imagen
    for j=1:N
        T(i,j)=(f(i,j)-minimo)*255/(maximo-minimo); % aplica la fórmula de cambio de rango a cada
pixel
        T(i,j)=round(T(i,j)); % Redondea el valor obtenido
    end
 end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        FUNCION UMBRALIZAR
% Calcula el nivel de gris que más se repite y establece dos umbrales
% sumándole y restándole 10. Los píxels que no llegan al umbral mínimo y
% que superan al maximo serán considerados bordes. Como entrada necesita la
% imagen generada por uno de los filtros prewitt y como salida dará una
% imagen binaria resultado de la umbralización, lista para marcar sobre la
% imagen original.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=umbralizar(prewitt)

histo=histograma(prewitt); % llama a la función histograma para obtener el vector histograma
de la imagen
grismax=maximoh(histo); % calcula el nivel de gris cuyo valor más veces se repite en la imagen
grismin=grismax-10; % establece el umbral inferior
grismax=grismax+10; % establece el umbral superior

[M,N]=size(prewitt); % se calculan las dimensiones de la imagen

for i=1:M
   for j=1:N % bucle anidado que recorre la imagen

      if prewitt(i,j)>grismax % si el nivel de gris supera el umbral superior se le asigna el color
blanco
         y(i,j)=255;
      else
         y(i,j)=0; % en caso contrario, el negro
       end

   end
end
```

```
for i=1:M

   for j=1:N % se vuelve a recorrer la imagen


      if prewitt(i,j)<grismin % si el nivel de gris es menor  que el umbral inferior, se le asigna el
color blanco

         y(i,j)=255;


      end

   end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        FUNCION HISTOGRAMA
% A partir de una imagen en escala de grises, calcula su histograma.
% Necesitará como entrada la imagen en escala de grises y como salida dará
% un vector de 256 valores correspondientes a los valores de gris 0 a 255
% indicando el número de píxels de la imagen en el que se repite cada
% valor.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function histo=histograma(f)

[M,N]=size(f); % calcula las dimensiones de la imagen
contador=0; % inicializa el contador

for n=1:256 % recorre los 256 niveles de gris diferentes

   for i=1:M % para cada nivel de gris recorre la imagen
      for j=1:N

         if f(i,j)==n-1  % cada vez que se nos encontremos con un pixel cuyo nivel de gris sea el
esperado,
                  % aumentaremos el contador en uno. Se le resta 1 al nivel de gris porque los
                  % niveles van de 0 a 255 y Matlab no puede trabajar con índices de tabla con
valor 0
                  % así que tenemos que trabajar con valores de 1 a 256
            contador=contador+1;

         end
      end
   end

   histo(n)=contador; % se guarda en el vector las veces que se repite el nivel de gris, indicado
por el contador
   contador=0; % se inicializa el contador
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    FUNCION MAXIMOH
% Calcula el nivel de gris que más se repite en una imagen. Para ello
% necesita como entrada el vector histograma de la imagen
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function y=maximoh(f);


a=max(f); % calcula el número de veces que se repite el nivel de gris que más veces se encuentra en la imagen


for i=1:256 % recorre el vector histograma
   if f(i)==a
       y=i; % cuando encuentre el nivel de gris correspondiente al maximo lo asignara a la salida
   end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       FUNCION MARCA
% A partir de la imagen original y una binaria correspondiente a uno de los
% filtros Prewitt, dará como salida una imagen marcada con verde la primera
% estimación de lo que son los lunares y marcas que buscamos.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function y=marca(orig,marca)


[M,N]=size(marca); % Calcula el tamaño de la imagen
y=orig; % realiza una copia de la imagen original


for i=1:M
   for j=1:N % recorre la imagen binaria


      if marca(i,j)==255 % si el pixel de la imagen binaria es blanco, se asigna al
correspondiente
                  % de la original el color verde (en RGB 0,255,0)
            y(i,j,1)=0;
            y(i,j,2)=255;
            y(i,j,1)=0;
       end
   end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       FUNCION PROCESADO2
%   Corrige marcas erróneas que se han obtenido tras un primer procesado.
%   Se ayudará de características como color y forma de las marcas para
%   poder desecharlas. El algoritmo llamará a otras subfunciones que se
%   encargarán de trabajos más específicos (eliminación del fondo, filtros para el vello,
%   bordes, etc.) Como variables de entrada precisará de la imagen
%   original y de la imagen marcada resultado del primer procesado. Dará
%   tres resultados diferentes: uno sin aplicar filtros, otro aplicando un
%   filtro para el vello y otro aplicando un filtro para los bordes. El
%   resultado óptimo dependerá de las características de la imagen. Los
%   tres resultados serán guardados en tres imágenes .bmp diferentes.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y= procesado2(marcada,original)


orig=original; % carga la imagen original y calcula sus dimensiones
[M,N]=size(orig);
N=N/3;
y=marcada; % Realiza una copia de la imagen marcada y la transforma para poder realizar
operaciones matemáticas.
y=double(y);


%Los siguientes bucles anidados, recorren los bordes de la imagen para borrar las
%marcas que han producido. En un marco de 6 píxels de grosor de la imagen,
%se asignará a la imagen marcada los valores de la original.
for i=1:6
   for j=1:N
      y(i,j,1)=orig(i,j,1);
      y(i,j,2)=orig(i,j,2);
      y(i,j,3)=orig(i,j,3);
   end
```

```
end
for i=M-6:M
   for j=1:N
      y(i,j,1)=orig(i,j,1);
      y(i,j,2)=orig(i,j,2);
      y(i,j,3)=orig(i,j,3);
   end
end
for i=1:M
   for j=1:6
      y(i,j,1)=orig(i,j,1);
      y(i,j,2)=orig(i,j,2);
      y(i,j,3)=orig(i,j,3);
   end
end
for i=1:M
   for j=N-6:N
      y(i,j,1)=orig(i,j,1);
      y(i,j,2)=orig(i,j,2);
      y(i,j,3)=orig(i,j,3);
   end
end

y=borra(y,orig); % Función que elimina marcas sobre zonas que no cumplen los requisitos de
color

fondo=piel(y);% Genera una imagen binaria diferenciando fondo y piel
for i=1:M %bucle que borra las marcas localizadas en el fondo
   for j=1:N
      if fondo(i,j)==255
         y(i,j,1)=orig(i,j,1);
         y(i,j,2)=orig(i,j,2);
         y(i,j,3)=orig(i,j,3);
```

```
        end
    end
end

filtro=filtro_pelo(orig,5,150); % Función que hace una estimación de las zonas de vello

y2=eliminar_pelo(y,orig,filtro); % Aplica el filtro para el vello


y=Extraccion_contornos(y);% realiza la extracción de contornos mediante operadores
morfológicos


orig=imread('original.bmp'); % Se marca sobre la imagen original y se guarda el primer de los
resultados (sin aplicar filtros)

for i=1:M
    for j=1:N
        if y(i,j)==255
            y1(i,j,1)=0;
            y1(i,j,2)=255;
            y1(i,j,3)=0;
        else
            y1(i,j,1)=orig(i,j,1);
            y1(i,j,2)=orig(i,j,2);
            y1(i,j,3)=orig(i,j,3);
        end
    end
end
y1=uint8(y1);
imwrite(y1,'Resultado_sin_filtros.bmp');
y1=double(y1);


y=estudio_contorno(y);% Elimina las marcas con formas que no pertenecen a lunares


orig=imread('original.bmp'); % Se marca sobre la imagen original y se guarda el resultado para
el filtro de bordes

for i=1:M
```

```
    for j=1:N
        if y(i,j)==255
            y1(i,j,1)=0;
            y1(i,j,2)=255;
            y1(i,j,3)=0;
        else
            y1(i,j,1)=orig(i,j,1);
            y1(i,j,2)=orig(i,j,2);
            y1(i,j,3)=orig(i,j,3);
        end
    end
end
y1=uint8(y1);
imwrite(y1,'Resultado_filtro_bordes.bmp');
y1=double(y1);

y2=Extraccion_contornos(y2);% se realiza la extracción para el filtro de vello

for i=1:M % Se marca sobre la imagen original y se guarda el resultado para el filtro de vello
    for j=1:N
        if y2(i,j)==255
            y1(i,j,1)=0;
            y1(i,j,2)=255;
            y1(i,j,3)=0;
        else
            y1(i,j,1)=orig(i,j,1);
            y1(i,j,2)=orig(i,j,2);
            y1(i,j,3)=orig(i,j,3);
        end
    end
end
y1=uint8(y1);
imwrite(y1,'Resultado_filtro_pelo.bmp');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%                    FUNCION BORRA

%   Elimina macas erróneas atendiendo a las propiedades de color de los  lunares.

%   Como Variables de entrada necesita la imagen original y la marcada

%   La salida será la imagen marcada, desechando los píxels que no cumplan

%   las características de color para tratarse de un lunar

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function y=borra(marcada,original)


[M,N]=size(original);

N=N/3;

y=marcada;

originalhsv=rgb2hsv(original);

original=double(original);

originalhsv=double(originalhsv);

%   En esta primera parte se calculan las dimensiones de la imagen y se crea una copia en
HSV



for i=1:M

   for j=1:N

      if y(i,j,1)<10 & y(i,j,2)>220 & y(i,j,3)<190 %Bucle anidado que recorre la imagen y revisa
que píxels están marcados


         r=original(i,j,1);

         g=original(i,j,2);

         b=original(i,j,3);

         gris=(r+g+b)/3;

         h=originalhsv(i,j,1);

         s=originalhsv(i,j,2)*100;

         v=originalhsv(i,j,3)*100;
```

```
        f=r-0.8*(g+b);


%    Se calculan las características de color para el pixel a analizar y se aplican las formulas para comparar en el siguiente paso si

%el pixel cumple las características obtenidas empíricamente


        if f>-9 & gris>37 & gris<123 & r>73 & r<169 & g>28 & g<116 & b<86 & b>9 & s<100 & s>23 & s<98 & v<92 & v>18 & (h<12 | h>344) %se cumplen las condiciones de color para lunares

            y(i,j,1)=marcada(i,j,1);

            y(i,j,2)=marcada(i,j,2);

            y(i,j,3)=marcada(i,j,3);

             %Si se cumplen el pixel seguirá manteniendo la marca

        else

            y(i,j,1)=original(i,j,1);

            y(i,j,2)=original(i,j,2);

            y(i,j,3)=original(i,j,3);

            %Si no se cumple, el pixel adquirirá su valor original

        end

    end

  end

end


y=uint8(y);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   FUNCION PIEL

%  Realiza una estimación de lo que es el fondo y  la superficie

%  correspondiente a la piel en la imagen. Generará una imagen binaria

%  donde los píxels blancos corresponderán al fondo y los negros a la

%  piel. Después se utilizará esta imagen para eliminar marcas que se

%  encuentren en la zona correspondiente al fondo. Como entrada necesitará

%  la imagen marcada.

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=piel(imagen);


%primero calcula las dimensiones de la imagen, la transforma a double y

%crea una copia en el dominio HSV para luego hacer las comparaciones.

[M,N]=size(imagen);

N=N/3;

ima=double(imagen);

imahsv=rgb2hsv(ima);




for i=1:M %bucles anidados que recorren la imagen

   for j=1:N

      if imagen(i,j,2)>250 %se comprueba si corresponde a un pixel verde

        y(i,j)=0; % en el caso de que sí sea, se estima como piel

        break %salta del bucle, con esto conseguimos agilizar la velocidad de ejecución del
programa

      end

     r=ima(i,j,1); % si el pixel no es verde, se extraen sus características de color

     g=ima(i,j,2);

     b=ima(i,j,3);

     gris=round((r+g+b)/3);

     s=imahsv(i,j,2);
```

```
        v=imahsv(i,j,3);
        if r>g & r>b & g+6>b & r>70 & g>40 & g<250 & b>25 & b<230 & gris>50 & gris<245 &
s<75 & v>25 % se comprueba si el pixel cumple las características de color
          y(i,j)=0; % en el caso de que sí, se considerará como piel
        else
          y(i,j)=255; % en el caso de que no, se considerará como fondo
        end
    end
end

% repetiremos la operación empezando desde las otras tres esquinas para
% conseguir un resultado óptimo
for i=1:M
    for j=N:-1:1
        if imagen(i,j,2)>250
            y(i,j)=0;
           break
        end
        r=ima(i,j,1);
        g=ima(i,j,2);
        b=ima(i,j,3);
        gris=round((r+g+b)/3);
        s=imahsv(i,j,2);
        v=imahsv(i,j,3);
        if not(r>g & r>b & g+6>b & r>70 & g>40 & g<250 & b>25 & b<230 & gris>50 & gris<245 &
s<75 & v>25 )
          y(i,j)=255;
      end
    end
end

for j=1:N
    for i=1:M
```

```
        if imagen(i,j,2)>250
            y(i,j)=0;
           break
        end
        r=ima(i,j,1);
        g=ima(i,j,2);
        b=ima(i,j,3);
        gris=round((r+g+b)/3);
        s=imahsv(i,j,2);
        v=imahsv(i,j,3);
      if not(r>g & r>b & g+6>b & r>70 & g>40 & g<250 & b>25 & b<230 & gris>50 & gris<245 &
s<75 & v>25 )
          y(i,j)=255;
      end
    end
end


for j=1:N
    for i=M:-1:1
        if imagen(i,j,2)>250
            y(i,j)=0;
          break
        end
        r=ima(i,j,1);
        g=ima(i,j,2);
        b=ima(i,j,3);
        gris=round((r+g+b)/3);
        s=imahsv(i,j,2);
        v=imahsv(i,j,3);
      if not(r>g & r>b & g+6>b & r>70 & g>40 & g<250 & b>25 & b<230 & gris>50 & gris<245 &
s<75 & v>25 )
          y(i,j)=255;
        end
```

```
    end
end

%realizaremos una dilatación para considerar como fondo los píxels
%correspondientes al borde entre el fondo i la piel de la imagen
y=dilatacion(y,10);
%tendremos una primera estimación de lo que es la piel




z=zeros(M,N);% se genera una imagen totalmente negra con las dimensiones de la original

%los siguientes dos bucles marcan como fondo los píxels correspondientes a
%los límites de la imagen
for i=1:N
    z(1,i)=255;
    z(M,i)=255;
end
for i=1:M
    z(i,1)=255;
    z(i,N)=255;
end


%recorreremos la imagen considerando como fondo los píxels que tengan como
%vecino al menos uno blanco y que también sean blancos en la primera
%estimación. Con esta técnica habremos eliminado de la primera estimación
%partes que se encuentran rodeadas de piel como puede ser el vello
for i=2:M-1
    for j=2:N-1
        if (z(i-1,j)==255 | z(i+1,j)==255 | z(i,j-1)==255 | z(i,j+1)==255)& y(i,j)==255 %
```

```
comprobamos si algún vecino es blanco y si en la primera estimación está incluido
            z(i,j)=255;% si se cumple consideraremos el pixel como fondo
        end
    end
end
z1=z;
% guardamos el resultado en Z1 y repetimos la operación empezando por las
% otras tres esquinas, generando Z2, Z3 y Z4

z=zeros(N,M);
for i=1:N
    z(i,1)=255;
    z(i,M)=255;
end
for i=1:M
    z(1,i)=255;
    z(N,i)=255;
end

for i=1:M
    for j=1:N
        y2(j,i)=y(i,j);
    end
end
for i=2:N-1
    for j=2:M-1
        if (z(i-1,j)==255 | z(i+1,j)==255 | z(i,j-1)==255 | z(i,j+1)==255)& y2(i,j)==255
            z(i,j)=255;
        end
    end
end
z2=z;
```

```
z=zeros(M,N);
for i=1:M
    z(i,1)=255;
    z(i,N)=255;
end
for i=1:N
    z(1,i)=255;
    z(M,i)=255;
end
for i=1:M-1
    for j=1:N-1
        y3(M-i,N-j)=y(i,j);
    end
end
for i=2:M-1
    for j=2:N-1
        if (z(i-1,j)==255 | z(i+1,j)==255 | z(i,j-1)==255 | z(i,j+1)==255)& y3(i,j)==255
            z(i,j)=255;
         end
    end
end
z3=z;
```

```
z=zeros(N,M);
for i=1:N
    z(i,1)=255;
    z(i,M)=255;
end
for i=1:M
    z(1,i)=255;
    z(N,i)=255;
end
for i=1:M-1
    for j=1:N-1
        y3(N-j,M-i)=y(i,j);
    end
end
for i=2:N-1
    for j=2:M-1
        if (z(i-1,j)==255 | z(i+1,j)==255 | z(i,j-1)==255 | z(i,j+1)==255)& y3(i,j)==255
            z(i,j)=255;
        end
    end
end
z4=z;



%Añadiremos los píxels considerados como fondo en las variables z1, z2, z3
%y z4 a la primera estimación
for i=1:M
    for j=1:N
        if z1(i,j)==255 | z2(j,i)==255 | z3(M+1-i,N+1-j)==255 | z4(N-j+1,M-i+1)==255
            y(i,j)=255;
        else
```

```
            y(i,j)=0;
        end
    end
end

%Realizaremos una dilatación para eliminar las marcas sobre los bordes de
%piel y fondo, obteniendo el resultado final.
y=dilatacion(y,11);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    FUNCION FILTRO_PELO
%   Realiza una estimación  del componente de vello en una imagen de la
%   piel. Generará una imagen binaria que posteriormente se utilizará para
%   desechar las marcas sobre el vello. Como entradas necesitará la imagen
%   original, el tamaño deseado para el filtro y a qué nivel de gris se va
%   a establecer el umbral.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y= filtro_pelo(imagen,tamano,umbral);


k=tamano^2-1; % Se calcula el número de píxels que va a tener el filtro (sin tener en cuenta el central)

tamano=floor(tamano/2); % Se calcula cuantos píxels vecinos respecto del central hay que sumar o restar para el cálculo del filtro


imagenbn=blanconegro(imagen); % se realiza una copia de la imagen en blanco y negro

[M,N]=size(imagenbn); % se realiza una copia de la imagen en blanco y negro

imagen=double(imagen); % se transforman las imágenes a double para poder realizar operaciones matemáticas

imagenbn=double(imagenbn);


for i=1:M

   for j=1:N % bucle anidado que recorre cada pixel de la imagen para calcular el filtro para cada uno de ellos

       suma=0; % se restablece la suma a 0

       for a=-tamano:tamano

          for b=-tamano:tamano % bucles anidados que recorren los píxels que entrarán en la máscara del filtro

             coord1=i+a;

             coord2=j+b; % Las coordenadas de cada pixel de la máscara se calculan mediante sumar las coordenadas parciales del pixel dentro de la máscara a las coordenadas globales del pixel central

             if a==0 & b==0 % si a y b son 0, se trata del pixel central, que en la máscara adquiere el valor de la suma del total de píxels en la máscara menos 1
```

```
                suma=suma+k*imagenbn(coord1,coord2); % se resta a la suma el valor para el
pixel central multiplicado por “k”

            elseif coord1>0 & coord1<M & coord2>0 & coord2<N % si la máscara no se sale del
rango de la imagen se suma el valor de pixel a la suma

                suma=suma-imagenbn(coord1,coord2);

            else % En el resto de casos la máscara se sale del rango de la imagen

                suma=0; % se le asigna a la suma el valor 0

            end

            y(i,j)=suma; % se guarda el valor de la suma para el pixel correspondiente en la
variable y

        end

    end

  end

end

y=uint8(y); % se transforma a uint8 porque no se van a realizar más operaciones matemáticas
```

```
for i=1:M

  for j=1:N % se recorre la imagen pixel a pixel

    if y(i,j)>umbral % si el pixel tiene un valor superior se le asignará el valor 255

      y(i,j)=255;

    else % si no supera el umbral se le asigna el valor 0, generando una imagen binaria

      y(i,j)=0;

    end

  end

end


 y=dilatacion(y,5); % se realiza la dilatación para incluir marcas finas como vello, manteniendo
las gruesas
```

```
%tendremos una primera estimación de vello, pero para obtener mejores
%resultados, tendremos en cuenta el grosor de las marcas generadas, ya
%que en el caso de que haya vello, se generará una "masa" cuya anchura será
%superior a la de los lunares.



for i=1:M
   for j=1:N %se recorre cada pixel de la imagen
      if y(i,j)==255 % si se trata de un pixel blanco pasaremos a realizar las siguientes
operaciones
         suma=0; % se inicializa la suma a 0
         for a=j:N  % se van comprobando si los píxels de las columnas siguientes son blancos
          if y(i,a)==255
             suma=suma+1; %por cada pixel blanco se suma uno a la variable
          else
             break % cuando encontremos un pixel negro saltaremos del bucle
          end
         end
         if suma<40 % comprobaremos si el grosor de la marca es inferior a 40, puede tratarse
de un lunar, así que no consideraremos estos píxels como vello
             y(i,j:a)=0;
         end
      end
   end
end


% se podría realizar el barrido desde otras direcciones, pero el resultado
% apenas mejora (además de complicar y ralentizar el procesado), por ello se opto por no
realizarlo.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        FUNCION ELIMINAR_PELO

%   Elimina los píxels correspondientes al vello en una imagen marcada. Una

%   imagen binaria indicará mediante los píxels blancos, que píxels son

%   tenidos en cuenta como vello. Esta imagen es el resultado de aplicar el

%   filtro para el vello. Como entradas necesitará la imagen original, la

%   imagen marcada y la imagen binaria

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function y=eliminar_pelo(marcada,original,binaria)

[M,N]=size(binaria); % Calcula el tamaño de la imagen


for i=1:M

   for j=1:N

      if binaria(i,j)==255 % Se recorre la imagen pixel a pixel con dos bucles anidados.

         y(i,j,1)=original(i,j,1); % si la binaria tiene el píxels correspondiente en blanco, se asigna
el valor de la imagen original

         y(i,j,2)=original(i,j,2);

         y(i,j,3)=original(i,j,3);

      else

         y(i,j,1)=marcada(i,j,1); % en caso contrario se le asigna el valor para la imagen marcada

         y(i,j,2)=marcada(i,j,2);

         y(i,j,3)=marcada(i,j,3);

      end

   end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        FUNCION EXTRACCION_CONTORNOS
%  A partir de una imagen marcada, resalta el contorno de la marca
%  prescindiendo de lo demás para poder observar el interior del lunar
%  y estudiar sus características. Como entrada precisa de la imagen
%  marcada y como salida dará la imagen con las marcas de los contornos
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=Extraccion_contornos(marcada)

[M,N]=size(marcada);
N=N/3;
marcada=double(marcada); % se calculan las dimensiones de la imagen marcada y se transforma a double

% recorreremos la imagen generando una imagen binaria con los píxels de las
% marcas en blanco y el resto en negro
for i=1:M
   for j=1:N
       if marcada(i,j,1)<10 & marcada(i,j,2)>200 & marcada(i,j,3)<190 % si se cumplen las condiciones de color de la marca
           y(i,j)=255; % asignamos el color blanco
       else
           y(i,j)=0; % Si no se cumplen el negro
       end
   end
end

y=dilatacion(y,7);
y=erosion(y,7); % se realiza un cierre (dilatación + erosión)para cerrar las marcas y no tener en cuenta solo los bordes

centro=erosion(y,3); % separaremos el centro de las marcas mediante erosionar el resultado
```

```
obtenido hasta ahora



for i=1:M
   for j=1:N % recorreremos la imagen pixel a pixel
      if y(i,j)==255 & centro(i,j)<240 % si el pixel pertenece a la imagen tras el cierre, pero no al centro será parte del contorno
         y(i,j)=255; % le asignaremos el color blanco
      else
         y(i,j)=0;  % en el caso de que no, le asignaremos el color negro generando una binaria
      end
   end
end


% obtendremos la imagen binaria correspondiente a los contornos. La
% guardaremos en unti8 y estará lista para marcar en la función procesado2
y=uint8(y);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                       FUNCION ESTUDIO_ CONTORNO
% Función que permite desechar marcas originadas por bordes o texturas que
% no pueden ser eliminadas por factores de color pero si de forma. Como
% entrada necesitará una imagen binaria a la que se ha aplicado la
% extracción de contornos. Y como salida dará una imagen binaria lista para
% marcar.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function y=estudio_contorno(binaria);


[M,N]=size(binaria); % Se calculan las dimensiones de la imagen
guardando=zeros(M,N); % crea una imagen negra con el tamaño de imagen original que hará de buffer
imwrite(guardando,'guardando.bmp'); % guardamos la imagen buffer


for i=1:M
   for j=1:N
      if binaria(i,j)==255 % recorre imagen binaria y en los píxels blancos llama a "cerrar contorno"
         binaria=cerrar_contorno(binaria,i,j);
      end
   end
end


 y=imread('guardando.bmp'); % carga la imagen buffer final para dar el resultado
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          FUNCION CERRAR_ CONTORNO
% Función que permite analizar los píxels pertenecientes al contorno de una
% marca para desechar según la forma de este contorno marcas que no pueden
% pertenecer a lunares. Como variable de entrada necesitará la imagen binaria
% correspondiente a los contornos señalados y las coordenadas de un pixel
% blanco, señalado previamente por la función "estudio_contorno". Como
% salida dará una imagen que va almacenando en un buffer y que al final
% será cargada en la función "estudio_contorno" y estará preparada para
% volver a marcar sobre la imagen original.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=cerrar_contorno(binaria,i,j);

%se crearan dos vectores, "Filas" donde iremos guardando la coordenadas de
%los píxels blancos para las filas y "Columnas" donde haremos lo propio para
%las columnas

Filas=i;
Columnas=j; %añadimos a los vectores las coordenadas del pixel inicial

Comprobar_fila=i;
Comprobar_columna=j; % guardamos las primeras coordenadas a comprobar, las correspondientes al pixel inicial



% generamos un bucle infinito, del que saltaremos con una instrucción "break" cuando uno de los píxels a
%analizar no tenga vecinos blancos
while 2>1

   while 2>1 %generamos otro bucle infinito
      if binaria(Comprobar_fila,Comprobar_columna+1)==255 % comprobamos si el pixel de la derecha es blanco
```

```
        Filas=[Filas Comprobar_fila]; %en el caso de que sea, añadimos sus coordenadas a los
vectores "Filas" y "Columnas"

        Columnas=[Columnas Comprobar_columna+1];

        Comprobar_columna=Comprobar_columna+1; %cambiaremos las coordenadas del pixel
a examinar

        binaria(Comprobar_fila,Comprobar_columna)=0; %marcaremos como negro el pixel para
no volver a analizarlo

        %volveremos al principio del bucle para ver si el siguiente pixel

        %tiene el vecino de la derecha blanco

      else

        break % si su vecino de la derecha no es blanco saldremos del bucle

      end

    end



    while 2>1 % repetiremos lo explicado para examinar el vecino superior-derecho

      if binaria(Comprobar_fila+1,Comprobar_columna+1)==255

        Filas=[Filas Comprobar_fila+1];

        Columnas=[Columnas Comprobar_columna+1];

        Comprobar_columna=Comprobar_columna+1;

        Comprobar_fila=Comprobar_fila+1;

        binaria(Comprobar_fila,Comprobar_columna)=0;

      else

        break

      end

    end



     while 2>1 %repetiremos para el vecino superior

      if binaria(Comprobar_fila+1,Comprobar_columna)==255

        Filas=[Filas Comprobar_fila+1];

        Columnas=[Columnas Comprobar_columna];

        Comprobar_fila=Comprobar_fila+1;

        binaria(Comprobar_fila,Comprobar_columna)=0;
```

```
  else
    break
  end
end

while 2>1 %repetiremos para el vecino superior izquierdo
  if binaria(Comprobar_fila+1,Comprobar_columna-1)==255
    Filas=[Filas Comprobar_fila+1];
    Columnas=[Columnas Comprobar_columna-1];
    Comprobar_fila=Comprobar_fila+1;
    Comprobar_columna=Comprobar_columna-1;
    binaria(Comprobar_fila,Comprobar_columna)=0;
  else
    break
  end
end

while 2>1 %repetiremos para el vecino de la izquierda
  if binaria(Comprobar_fila,Comprobar_columna-1)==255
    Filas=[Filas Comprobar_fila];
    Columnas=[Columnas Comprobar_columna-1];
    Comprobar_columna=Comprobar_columna-1;
    binaria(Comprobar_fila,Comprobar_columna)=0;
  else
    break
  end
end

 while 2>1 %repetiremos para el vecino inferior-izquierdo
  if binaria(Comprobar_fila-1,Comprobar_columna-1)==255
    Filas=[Filas Comprobar_fila-1];
    Columnas=[Columnas Comprobar_columna-1];
    Comprobar_fila=Comprobar_fila-1;
```

```
        Comprobar_columna=Comprobar_columna-1;
        binaria(Comprobar_fila,Comprobar_columna)=0;
      else
        break
      end
    end

    while 2>1 %repetiremos para el vecino inferior
      if binaria(Comprobar_fila-1,Comprobar_columna)==255
        Filas=[Filas Comprobar_fila-1];
        Columnas=[Columnas Comprobar_columna];
        Comprobar_fila=Comprobar_fila-1;
        binaria(Comprobar_fila,Comprobar_columna)=0;
      else
        break
      end
    end

      while 2>1 %repetiremos para el vecino inferior derecho
      if binaria(Comprobar_fila-1,Comprobar_columna+1)==255
        Filas=[Filas Comprobar_fila-1];
        Columnas=[Columnas Comprobar_columna+1];
        Comprobar_fila=Comprobar_fila-1;
        Comprobar_columna=Comprobar_columna+1;
        binaria(Comprobar_fila,Comprobar_columna)=0;
      else
        break
      end
    end


      if (Comprobar_fila==i & Comprobar_columna==j)|
  (binaria(Comprobar_fila,Comprobar_columna+1)~=255 &
```

```
binaria(Comprobar_fila+1,Comprobar_columna)~=255 &
binaria(Comprobar_fila+1,Comprobar_columna+1)~=255 &
binaria(Comprobar_fila,Comprobar_columna-1)~=255 & binaria(Comprobar_fila-
1,Comprobar_columna)~=255 & binaria(Comprobar_fila-1,Comprobar_columna-1)~=255 &
binaria(Comprobar_fila-1,Comprobar_columna+1)~=255 &
binaria(Comprobar_fila+1,Comprobar_columna-1)~=255)%si vecinos no son blancos

      break % si el pixel a analizar no tiene vecinos blancos saltaremos del bucle infinito inicial.
Si no, volveremos a comprobar los vecinos de todas direcciones de nuevo

   end

end


% en este punto habremos recorrido todo el contorno y tendremos las

% coordenadas de cada pixel del contorno almacenadas en los vectores

% "Filas" y "Columnas" preparados para realizar los cálculos necesarios




[W,L]=size(Filas); % Calculamos las dimensiones del vector filas. En la variable L se guardará
el número de píxels incluidos


maxf=max(Filas); % Se calcula la fila máxima y mínima

minf=min(Filas);

maxc=max(Columnas); % Se calcula la columna máxima y mínima

minc=min(Columnas);

Lado1=maxf-minf; % Obtenemos la anchura y la altura del lunar

Lado2=maxc-minc;

Dif=abs(Lado1-Lado2); % Calculamos la diferencia entre anchura y altura


fila1=Filas(1); % Calculamos las coordenadas para el primer pixel

columna1=Columnas(1);

fila2=Filas(L); % Calculamos las coordenadas para el ultimo pixel

columna2=Columnas(L);

modulo=round(((fila1-fila2)^2+(columna1-columna2)^2)^0.5); % calculamos el módulo

guardando=imread('guardando.bmp'); % Cargamos la imagen almacenada en el buffer
```

```
if modulo<10 & Dif<5 %Si el lunar cumple con los requisitos de diferencia entre anchura y altura
además de distancia entre píxels inicial y final lo añadiremos a la imagen buffer

   for i=1:L

      f=Filas(i);

      c=Columnas(i);

      guardando(f,c)=255;

   end

imwrite(guardando,'guardando.bmp'); %guardaremos la imagen buffer

end


y=binaria; %restauraremos la imagen como estaba al principio y saldremos de la función. Los
resultados son guardados en la imagen buffer.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                       FUNCION DILATACION
%   Realiza la operación morfológica dilatación sobre una imagen binaria.
%   Como entrada precisará de la imagen binaria y del tamaño del elemento
%   estructurante.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function y=dilatacion(f,n)


f=double(f); % se transforma la imagen a double y se calcula su tamaño
[M N]=size(f);
y=zeros(M,N); % se genera una imagen negra con las mismas dimensiones


mitad=floor(n/2); % se calcula el numero de píxels que habrá que recorrer hacia delante
                  %o hacia atrás para abarcar el elemento estructurante


for i=1:M
   for j=1:N % se recorre pixel a pixel la imagen binaria


      if f(i,j)>250 & f(i,j)<260 % si un pixel es blanco


         for a=-mitad:mitad
            for b=-mitad:mitad % se recorre el elemento estructurante centrado en el pixel


               if 0<i+a & i+a<M & 0<j+b & j+b<N % se comprueba que no coincida con las
esquinas de la imagen
                  y(i+a,j+b)=255; % se asigna el blanco a los píxels que coincidan con el elemento
estructurante
               end


            end
```

```
            end
        end


    end
 end
f=uint8(f);
y=uint8(y); % se transforman las imágenes binarias a uint8
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       FUNCION EROSION
%   Realiza la operación morfológica erosión sobre una imagen binaria.
%   Como entrada precisará de la imagen binaria y del tamaño del elemento
%   estructurante.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=erosion(f,n)



f=double(f); % se transforma la imagen a double
[M N]=size(f); % se calcula su tamaño
y=zeros(M,N); % se genera una imagen negra del mismo tamaño
mitad=floor(n/2);% se calcula el numero de píxels que habrá que recorrer hacia delante
                 %o hacia atrás para abarcar el elemento estructurante
n=n^2; % se calcula el numero de píxels que tiene el estructurante




for i=1:M
   for j=1:N % se realiza un barrido de la imagen


      suma=0; % se inicializa la suma a 0


      for a=-mitad:mitad
         for b=-mitad:mitad % se recorre el elemento estructurante


            if 0<i+a & i+a<M & 0<j+b & j+b<N % si las coordenadas no se salen de los límites de
la imagen


               suma=suma+f(i+a,j+b); % se suman los valores de los píxels que abarca el
elemento
```

```
            end


          end
        end


      if suma>250*n & suma<260*n % si la suma coincide con el numero de píxels del elemento
por 255 (aproximando)
         y(i,j)=255; % Se le asigna al pixel el valor 255
      end


   end
end


f=uint8(f);
y=uint8(y); % se transforman las imágenes a uint8
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       FUNCION PROCESANDO
%  Mostrará la imagen original en una ventana mientras se realiza todo el
%  procesado
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function varargout = procesando(varargin)


%Parte del programa generada automáticamente por Matlab
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @procesando_OpeningFcn, ...
                   'gui_OutputFcn',  @procesando_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function procesando_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);


%Parte del programa editada por el usuario
function varargout = procesando_OutputFcn(hObject, eventdata, handles)
```

```
varargout{1} = handles.output;


foto=imread('original.bmp'); %carga la imagen original
imshow(foto); %muestra la imagen original en una ventana
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       FUNCION EXITO
%   Se ejecutara al final del procesado y su función será la de informar
%   al usuario que ha terminado el procesado y de permitirle comparar los
%   diferentes resultados con la imagen original. También permitirá volver
%   a la pantalla inicial para introducir una nueva imagen a procesar
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = guidetemplate0(varargin)

%Parte del programa generada automáticamente por Matlab
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @guidetemplate0_OpeningFcn, ...
                   'gui_OutputFcn',  @guidetemplate0_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function guidetemplate0_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = guidetemplate0_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

```
%Parte del programa editada por el usuario



function pushbutton1_Callback(hObject, eventdata, handles) %Botón que mostrará la imagen original al pinchar sobre él.

fig=imread('original.bmp');

figure;

imshow(fig);



function pushbutton2_Callback(hObject, eventdata, handles) %Botón que mostrará el resultado sin utilizar filtros

fig=imread('Resultado_sin_filtros.bmp');

figure;

imshow(fig);



function pushbutton3_Callback(hObject, eventdata, handles) %Botón que mostrará el resultado sin utilizando un filtro para el vello

fig=imread('Resultado_filtro_pelo.bmp');

figure;

imshow(fig);



function pushbutton4_Callback(hObject, eventdata, handles) %Botón que cerrará el programa 'exito' y volverá a la pantalla inicial

interfaz

close(Exito)
```

```
function pushbutton5_Callback(hObject, eventdata, handles) %Botón que mostrará el resultado utilizando un filtro para bordes

fig=imread('Resultado_filtro_bordes.bmp');

figure;

imshow(fig);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%                       FUNCION ERROR_EN_IMAGEN
%    Cuando se produzca un error o la imagen no se ha introducido
%    correctamente, mostrará un aviso al usuario y emitirá una señal
%    acústica
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function varargout = Error_en_imagen(varargin)


%Parte de la función generada por Matlab
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Error_en_imagen_OpeningFcn, ...
                   'gui_OutputFcn',  @Error_en_imagen_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end


if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function Error_en_imagen_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = Error_en_imagen_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

```
%Se mostrará la ventana de error


%Parte editada por el usuario
alert=wavread('Error.wav'); %da señal acústica para decir que termina
sound(alert);
```