# 3. Desarrollo de *Cryojab 2.0*[3]

Como se ha comentado con anterioridad, para el desarrollo de esta interfaz de usuario se ha elegido un lenguaje de programación web dinámico para implementar la solución adoptada.

Para el desarrollo de la aplicación, tras sopesar el uso de un lenguaje compilado o un lenguaje interpretado, se ha optado por esta segunda opción. Y es que la solución ideada requeriría que el programa modificase su estructura en función de los datos que el usuario fuese introduciendo a medida que avanzase a través de las etapas establecidas. Y esto no era posible con un lenguaje compilado (no al menos sin complicar en exceso la programación de la aplicación).

El lenguaje elegido no es otro que ***PHP*** (*PHP Hypertext Pre-procesor*) es un lenguaje de programación usado frecuentemente para la creación de contenido para sitios webs. Como hemos dicho, es un lenguaje interpretado usado para la creación de aplicaciones para servidores, o creación de contenido dinámico para sitios web. Y es este último uso el que más nos interesa. Además, el fácil uso y la similitud con lenguajes de programación estructurada más comunes como ***C*** o ***Perl***, permiten que la mayoría de los programadores experimentados creen aplicaciones complejas con una curva de aprendizaje muy suave.

El código ***PHP*** lo encontramos habitualmente embebido en código ***HTML***, formando parte de las páginas webs que visitamos con nuestros navegadores (no confundir con el código de la página que se nos muestra en el navegador. ***PHP*** es un lenguaje interpretado, y como causa de esa interpretación el servidor ***PHP*** genera una

[3] Ref.: [8], [16], [17], [18], [24], [25], [27]

salida en forma de código ***HTML*** que es lo que llega al navegador y es lo que se muestra al usuario).

Independientemente de las ventajas del propio lenguaje en sí, aparece una nueva posibilidad muy importante. Y es la capacidad de ejecutar la aplicación vía web, es decir, de manera remota. Con vistas a instalar el sistema en un entorno (cerrado o abierto), se puede configurar un único equipo que esté accesible a todos los de la red, pudiendo realizar simulaciones sin necesidad de instalar ningún tipo de software en estos equipos.

## 3.1. Introducción a *HTML*

***HTML*** son las siglas de *HyperText Mark-up Language*. ***HTML*** fue definido a principios de 1990 por *Tim Berners-Lee* como un subconjunto del conocido ***SGML*** (*Standard Generalized Markup Language*, sirve para especificar las reglas de etiquetado de documentos y no impone en si ningún conjunto de etiquetas en especial) y crea algo más valioso aun, el *World Wide Web*. En 1991, *Tim Berners-Lee* crea el primer navegador de ***HTML*** que funcionaba en modo texto bajo el entorno ***UNIX***.

Desarrollar cogido ***HTML*** no requiere ningún tipo de editor específico. Con cualquier editor de texto básico podemos escribir nuestras páginas.

El lenguaje ***HTML*** utiliza una serie de etiquetas o marcas, que consisten en breves instrucciones de comienzo y final, mediante las cuales se determinan la forma en la que debe aparecer en el navegador el texto, así como las imágenes y demás elementos, en la pantalla del ordenador.

Toda etiqueta se identifica porque está encerrada entre los signos de menor que y mayor que (< >), y algunas tienen atributos que pueden tomar algún valor. Existen tres etiquetas básicas que siempre deben aparecer en una página web: *html*, *head* y *body*. Y a partir de estas 3 se desarrolla el resto del código.

## 3.2. Introducción a *PHP*

### 3.2.1. Historia de *PHP*

***PHP*** fue diseñado originalmente en ***PERL***, seguidos por la escritura de un grupo de ***CGI*** binarios escritos en lenguaje ***C*** por el programador danés-canadiense *Rasmus Lerdorf* en 1994 para mostrar su currículum vitae y guardar ciertos datos, como la cantidad de tráfico que su página web recibía.

Posteriormente, dos programadores israelíes del *Technion* (Instituto Tecnológico Israelí), *Zeev Suraski* y *Andi Gutmans*, reescribieron el analizador sintáctico (*parser*) en 1997 y crearon la base del ***PHP3***, cambiando el nombre del lenguaje a la forma actual. Inmediatamente comenzaron experimentaciones públicas de ***PHP3*** y fue publicado oficialmente en junio de 1998. Y así ha ido evolucionando hasta encontrarnos hoy en día con ***PHP5***, con ***PHP6*** en camino.

Para aquellos programadores que ya tengan experiencia con el lenguaje de programación C, comprender la sintaxis de PHP les resultara extremadamente sencillo. Las dos únicas diferencias reseñables es que no hay que instanciar variables al comienzo de la función (se pueden crear en cualquier parte), y que las variables llevan delante del nombre el símbolo del dólar ($).

### 3.2.2. Usos de *PHP*

Los principales uso de ***PHP*** son:

- Programación de páginas web dinámicas, habitualmente en combinación con el motor de bases de datos ***MySQL***, aunque cuenta con soporte nativo para otros motores, incluyendo el estándar ***ODBC*** (*Open Database Connectivity*), lo que amplia en gran medida sus posibilidades de conectividad.
- Programación en consola, al estilo de ***Perl*** o ***Shell scripting***.
- Creación de aplicaciones graficas independientes del navegador, por medio de la combinación de ***PHP*** y ***Qt/GTK+***, lo que permite desarrollar aplicaciones de escritorio en los sistemas operativos en los que está soportado.

### 3.2.3. Ventajas de *PHP*

Las ventajas de ***PHP*** son:

- Es un lenguaje multiplataforma, es decir, que puede ser ejecutado en diversas arquitecturas (***Windows***, ***GNU/Linux***, ***Mac OS X***…).
- Capacidad de conexión con la mayoría de los manejadores de base de datos que se utiliza en la actualidad, destacando su conectividad con ***MySQL***.
- Leer y manipular datos desde diversas fuentes, incluyendo datos que pueden ingresar los usuarios desde formularios ***HTML***.

- Capacidad de expandir su potencial utilizando la enorme cantidad de módulos o extensiones que se han ido desarrollando con el tiempo.
- Posee una amplia documentación en su página web oficial, www.php.net, entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite las técnicas de *Programación Orientada a Objetos*.
- Permite crear formularios para páginas web.
- Posee una biblioteca nativa de funciones sumamente amplia.
- No requiere definición de tipos de variables ni manejo detallado a bajo nivel.

## 3.3. Solución de los problemas del modulo principal

El primer problema a abordar debía ser los errores que el código original poseía, y que no permitían un correcto funcionamiento de la aplicación.

Cuando se ejecutaba la aplicación para realizar una simulación, independientemente de los parámetros de entrada, el programa ejecutaba todas sus rutinas correctamente. Pero al finalizar, habiendo creado correctamente todos los ficheros con los resultados de la simulación, la aplicación devolvía una serie de errores. Errores que no afectaban al discurrir de la misma, pero que, obviamente, debían ser subsanados, pues en un futuro podrían acarrear más de un problema.

Tras un periodo de inspección del código original, y tomando como punto de partida que los archivos con los resultados de la simulación se generaban de la forma esperada, se llegó a la conclusión de que el problema se debía encontrar en la parte en la que la aplicación liberaba la memoria reservada para almacenar temporalmente los datos de la simulación. Esta zona se encuentra al final del código de ***main.cpp***.

Después de visualizar paso a paso la simulación mediante el depurador, se encontró el problema. En efecto, el problema se encontraba en la zona de liberación de memoria. Concretamente la zona de memoria a la que apuntaba el puntero "*cae*" de tipo "*char*" no era liberada. En dicha zona de memoria se almacenaba los nombres de los distintos ficheros de tipo 0. La instrucción concreta que había que añadir al código es la que sigue:

```
delete[] cae;
```

Una vez introducida en el código en la función principal, la aplicación, nuevamente compilada, realiza simulaciones sin devolver ningún tipo de errores.

## 3.4. Nueva implementación de *Cryojab-color*

Como se comentó anteriormente, uno de los puntos críticos de la primera versión de ***Cryojab*** es que no realizaba adecuadamente la detección de los colores en la imagen seleccionada para realizar la simulación. Esto convertía a ***Cryojab*** es una aplicación inútil, pues el trabajo de realizar los ficheros de tipo 1 de forma manual es un proceso demasiado farragoso. La solución: rediseñar ***Cryojab-color***.

En un primer momento se pensó usar el mismo lenguaje que ***Cryojab***, pero la nueva implementación no es tan sencilla como la anterior, y realiza un proceso que puede alargarse en el tiempo. Así que se opto por migrar la aplicación a código **PHP**. ¿Qué ventajas supone esto? Mayor velocidad de ejecución y mayor capacidad de proceso (no tiene que reservar memoria de manera dinámica, sino que se va utilizando a medida que se necesita, eliminando los márgenes de memoria establecidos en un código compilado). Es esto lo que confiere a ***Cryojab 2.0*** de una gran capacidad de trabajo. Ahora se pueden generar los ficheros tipo 1 de cualquier imagen en un tiempo record.

El gran problema de la versión anterior de ***Cryojab-color*** es que no detectaba adecuadamente los colores. Esto es porque las pretensiones eran demasiado elevadas. La idea era poder detectar cualquier color y asignarlo a un material de forma automática, sin indicar a la aplicación ninguna directriz. Esto es imposible. Es decir, un programa no puede decidir que, por ejemplo, el rojo corresponde con las células o el blanco con el tejido conjuntivo. Esto hay que indicárselo a la aplicación de alguna manera. Todo se complica más debido a la estrecha relación que existe entre los ficheros generados por ***Cryojab-color*** el fichero de tipo 2, donde se almacenan los parámetros de los materiales contenidos en la fotografía. Así pues, para esta nueva implementación se ha optado por separar esto en dos pasos. Primero detectaremos los colores de la imagen, pero almacenaremos de alguna manera los colores seleccionados para, posteriormente, asignarle los materiales correspondientes.

También se comento que ***Cryojab-color*** era incapaz de reaccionar ante un color que no espere. Es decir, si la aplicación tiene como orden encontrar una serie de colores y cuando está examinando un pixel decide que el color del mismo se corresponde con otro color que no es ninguno de los esperados, la aplicación se volvía inestable.

Ante estos dos problemas, la idea que surge es la de hacer una prelectura de la imagen para realizar una aproximación. Y no es otra que cuando ***Cryojab-color*** encuentre un color que no espera lo aproxime al color mayoritario de la imagen. Así conseguimos una mayor similitud con la imagen original, aunque obviamente perdemos información. Pero al fin y al cabo, es una información que no nos es útil, pues no contemplamos esos colores a la hora de realizar la simulación, de manera que si son sustituidos por el color mayoritario, el error cometido es el menor posible. Obviamente la prelectura que se realiza solo contempla los colores que hayamos seleccionado. Por eso es muy importante haber realizado un tratamiento previo de la fotografía que introduzcamos en la aplicación, pues si los materiales que nos interesa estudiar no se corresponden con los colores que podemos seleccionar, estos serán descartados por la aplicación al entender que no son ninguno de los colores que espera encontrar dentro de la imagen.

A ***Cryojab-color*** hay que informarle de que colores queremos que tenga en cuenta. Para ello, la nueva versión de ***Cryojab-color*** recibe una serie de parámetros muy concretos. En primer lugar un fichero con los colores que debe contemplar. A cada color se le asignara un código. Si el color vale -1 indica que no se ha seleccionado. Si no es un número negativo, se indica que el color ha sido seleccionado, y el número que contenga es el código que le identifica a la hora de generar los ficheros de tipo 1. Además hay que indicarle el número de nodos que contiene la subestructura. Para esta implementación se ha aproximado que todas las subestructuras serán cuadradas y tendrán el mismo número de nodos.

En la versión anterior de ***Cryojab-color*** hacíamos uso del software ***GIMP*** para el paso a formato textual. Pero esto lo tenía que realizar el usuario. No se ejecutaba de

forma automática. Para esta nueva versión se ha decidido utilizar un software distinto: ***ImageMagick***. ***ImageMagick*** es un paquete que sirve para editar, crear y componer imágenes, pudiendo leer, convertir y escribir imágenes en una infinidad de formatos. Además incluye una gran cantidad de *API*'s (*Application Programming Interface*) para casi todos los lenguajes de programación usados hoy en día.

De todas formas nosotros utilizaremos solo una de las aplicaciones que forma el paquete. Y no es otra que ***convert***. La aplicación ***convert.exe*** no requiere instalación y puede ser ejecutada mediante línea de comandos. Y es esta última característica la que más nos interesa, pues podemos realizar llamadas a ejecución en línea de comandos mediante código PHP. Mediante una simple llamada a ***convert*** podemos redimensionar y cambiar de formato la imagen para adecuarla a lo que nuestro sistema requiere.

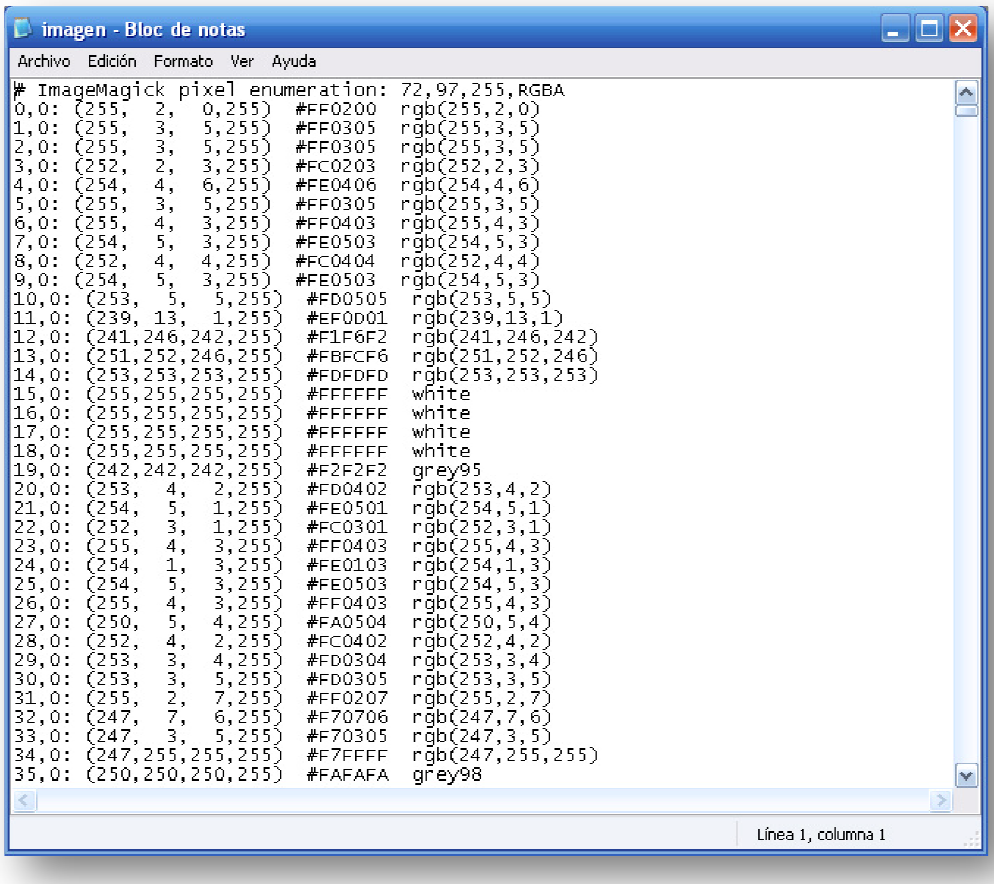



Fig. 3.1 – Imagen en formato textual generado por *ImageMagick*

***ImageMagick*** realiza una conversión a formato textual mucho más rica de lo que lo hacia ***GIMP***. Este último nos devolvía un fichero en forma de matriz en la que 3 elementos indicaban las componentes RGB de cada pixel situados espacialmente en el fichero de texto en la misma posición en la que se encontrarían en la imagen. Esto convertía la lectura del fichero en un proceso muy tedioso. Sin embargo ***ImageMagick*** genera un fichero que ofrece más información y de forma mucho más estructurada que ***GIMP***.

Como vemos en la figura 3.1, nos ofrece las coordenadas del pixel, el color desglosado en sus componentes RGB, su traducción a hexadecimal y la interpretación que realiza ***ImageMagick*** de dicho color. Resulta mucho más cómodo procesar este tipo de ficheros. Además abre la puerta a una posible reconstrucción de una imagen.

Entonces, a este fichero se le realiza una primera prelectura para identificar los colores que hayamos seleccionado. Pero, ¿Cómo interpretamos estos colores? Existen 65.536 posibles combinaciones. La solución adoptada no es otra que acotar los colores que se pueden seleccionar y aproximar a dichos colores. Así el usuario puede seleccionar 8 posibles colores: blanco, negro, rojo, azul, verde, amarillo, magenta y cian. Los códigos RGB de estos 8 colores son los más significativos pues llevan cada componente a su máximo o a su mínimo. No hay valores intermedios. Entonces, para realizar la identificación definimos un umbral (aproximadamente en la mitad del intervalo de valores que puede tomar una componente RGB, es decir, si una componente RGB puede tomar valores de 0 a 255, definiremos un umbral en torno a 125). Entonces, para cada componente RGB comprobamos si su valor se encuentra por encima o por debajo del umbral. Entonces realizamos una comprobación de con cuál de las 8 posibilidades corresponde.

De esta manera realizamos una primera prelectura para buscar el color mayoritario dentro de la imagen.

Tras esta prelectura pasamos a identificar qué color contiene cada pixel. La identificación se realiza de la misma manera que en la prelectura, pero en este caso generamos una matriz de dos dimensiones en la que almacenamos el color detectado (en caso de que no corresponda con ningún color seleccionado se almacena el color predominante). Las coordenadas del *array* son las coordenadas de los pixeles, y el valor que se almacena es el código del color detectado.

Una vez obtenida la matriz con la imagen completa muestreada, generamos un nuevo fichero que, mediante ***ImageMagick***, nos devolverá la imagen que ***Cryojab-color*** ha detectado. Así podemos comparar si la detección ha sido adecuada antes de realizar cualquier simulación.

Finalmente generamos los ficheros tipo 1 en función del tamaño de la imagen y del número de nodos que contendrá cada subestructura (pasado como parámetro).

## 3.5. Definición e implementación de las etapas

El objetivo principal de esta nueva versión de ***Cryojab*** es facilitar al usuario la realización de simulaciones. Para ello se va a diseñar una interfaz que permita al usuario introducir los datos de la manera más cómoda posible, con todo tipo de información, y de manera que no pueda introducir datos erróneos que lleven a un mal funcionamiento de la aplicación original. Posteriormente, se realizara un tratamiento de la información

obtenida en forma de ficheros para mostrar gráficamente los resultados, y poder extraer datos concretos de manera sencilla sin tener que editar ficheros.

Así que se irán definiendo una serie de pasos que se implementaran posteriormente en código PHP embebido en HTML.

Para tener un sistema de directorios más ordenado y práctico se ha optado por separar la parte ejecutable, es decir, ***Cryojab***, de la interfaz grafica. Todos los ficheros que se creen se almacenaran en la carpeta donde se encuentre el modulo principal de ***Cryojab***.

### 3.5.1. Etapa de bienvenida

En primer lugar se le mostrará al usuario la ventana de presentación de la aplicación Web. En ella se le hará una serie de recomendaciones para llegar a realizar una simulación con éxito.

Mas concretamente se le comunica al usuario que la imagen que va a procesar debe haber sido tratada digitalmente con anterioridad para una identificación más o menos exacta de los materiales con los colores que corresponda.

***Cryojab 2.0*** podría incluirse en el grupo de los servicios webs, por ser una aplicación que se sirve a través de servidores HTTP. Es necesario el uso de un navegador para poder realizar simulaciones.


Fig. 3.2 – Captura de la Etapa de Bienvenida

Los exploradores poseen controles de navegación propios. En esta aplicación se recomienda no usar dichos controles, porque cada etapa genera una serie de archivos que deben ser eliminados si se vuelve atrás en la sucesión de pasos. Para ello se han dispuesto una serie de controles de navegación propios de la aplicación, que se encargan de limpiar los datos introducidos en el paso anterior.

Además de lo anteriormente dicho, en este paso 0 se realiza una limpieza de los ficheros que pudiesen haber quedado almacenados en el directorio donde se encuentra ubicado ***Cryojab*** debidos a una simulación anterior.

### 3.5.2. Etapa I. Carga de la imagen

Éste es realmente el primer paso del proceso que se debe llevar a cabo para realizar una simulación. Se le muestra al usuario un único campo de formulario con explorador para que pueda cargar la imagen que desea estudiar.

Debemos volver a recordar que esta imagen ha debido ser tratada previamente para una identificación correcta de los colores. Más adelante se explicara cómo se realiza esta detección y las limitaciones que posee.

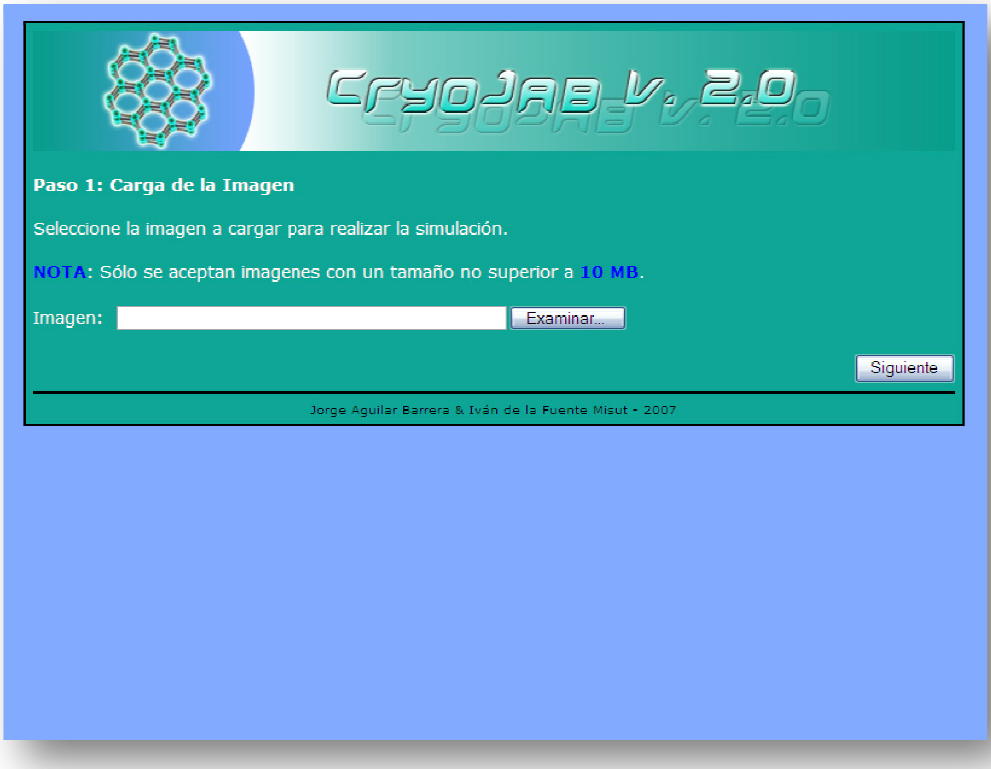



Fig. 3.3 – Captura de la Etapa I

Se realizan una serie de comprobaciones con la ruta introducida para evitar errores. En primer lugar se comprueba si se ha introducido algo en el campo. En caso negativo se muestra un error por pantalla.

Posteriormente se comprueba si el fichero que se pasa es una imagen. En caso negativo se muestra un error por pantalla. Si la imagen ha sido cargada adecuadamente se pasa a la etapa siguiente.

Éste es el primer paso en el que se pone a disposición del usuario la posibilidad de volver atrás en el proceso. Cada vez que pulsemos este botón a lo largo de la aplicación se deshará lo realizado en el paso anterior. En este caso se eliminaría la imagen cargada y se liberarían las variables de paso de parámetros al siguiente paso.

### 3.5.3. Etapa II. Dimensionamiento de la imagen

Una vez cargada la imagen en la aplicación se le ofrece al usuario la posibilidad de redimensionar la fotografía para ajustar el tamaño y conseguir una simulación optima.

Como el sistema no puede controlar el tamaño original de la imagen cargada, lo primero que hace la aplicación es obtener las dimensiones de la fotografía y, si son excesivas, se muestra ajustada a un tope (pero sin redimensionar aun). El usuario tiene la opción de cambiar dicha resolución introduciendo el nuevo ancho. Solo se introduce el ancho porque el sistema está diseñado de manera que no introduce distorsión, manteniendo las proporciones.

Se le comunica al usuario que no puede introducir una resolución igual a cero, pues nos quedaríamos sin imagen. Al dato introducido por el usuario se le realizan una serie de controles. Se comprueba si es un dato numérico, distinto de cero y positivo. En el caso de que esto no se cumpla se mostraría un mensaje de error por pantalla.


Fig. 3.4 – Captura de la Etapa II

### 3.5.4. Etapa III. Comprobación de dimensionamiento

En esta etapa se solicita la aprobación del usuario tras el redimensionamiento de la fotografía. Se le muestra la imagen con la nueva resolución. Si el usuario está conforme con esta nueva resolución y avanza, la aplicación simplemente realiza la transformación a formato textual de la imagen para su posterior procesado.

Si el usuario vuelve atrás, se elimina la imagen redimensionada y se vuelve a mostrar la pantalla de redimensionamiento con la fotografía original.

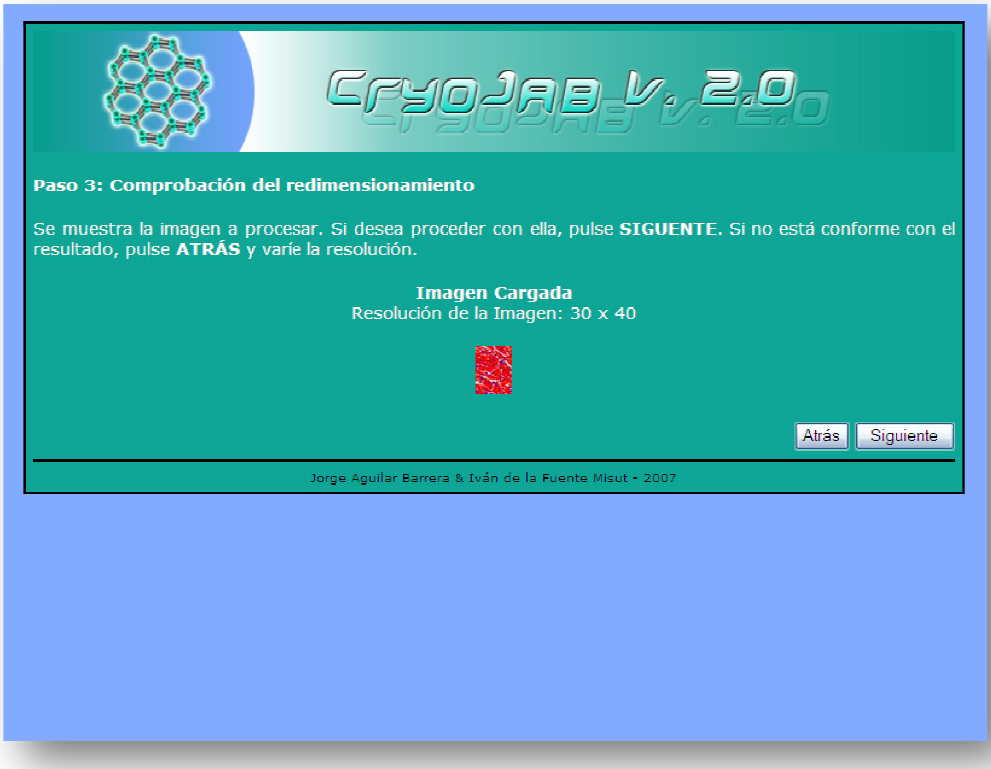



Fig. 3.5 – Captura de la Etapa III

### 3.5.5. Etapa IV. Selección de los colores

Ésta es una de las etapas más complejas que nos encontraremos a lo largo del proceso. Consiste en seleccionar los colores que se encuentran en la foto y que más tarde asignaremos a los materiales que componen el órgano que queremos estudiar. En este paso es de vital importancia que la imagen sea claramente identificable. Y es que este es el primer paso donde se generan ficheros que recibirá ***Cryojab*** como entrada.

En primer lugar se comprueba si se ha seleccionado al menos un color, así como si el número de nodos en la subestructura es un valor numérico positivo distinto de cero. En caso contrario se muestra un error por pantalla. Si los datos son correctos, llamamos a la función fichero1. Esta función genera el fichero con el código de colores que sirve de entrada para ***Cryojab-color***. Se realiza la llamada a ***Cryojab-color*** y, tras su ejecución, se reconvierte el fichero con la imagen muestreada generada por ***Cryojab-color*** en formato textual.


Fig. 3.6 – Captura de la Etapa IV

## 3.5.6. Etapa V. Comprobación de imagen muestreada

En este paso se muestran en pantalla la imagen original (redimensionada o no) y la imagen muestreada tras el proceso de creación de los ficheros 1.

Este paso proporciona al usuario la imagen con la que se va a llevar a cabo la simulación. Es deber del usuario decidir si la imagen muestreada resultante es lo suficientemente buena como para realizar la simulación que desea llevar a cabo.

Si el usuario está de acuerdo con la imagen obtenida tras el muestreo, prosigue adelante. Pero si no se ha obtenido una imagen adecuada, debe volver atrás y variar los parámetros introducidos hasta ahora (resolución, colores…) o tal vez debe volver a editar la fotografía para conseguir una mayor identificación de los colores.

Este paso es realmente importante, pues sin él el usuario se lanzaría a realizar una simulación a ciegas y obtendría resultados sin saber realmente cual es la entrada que ha recibido el sistema. Con esta etapa el usuario sabe a ciencia cierta qué es lo que va a simular.


Fig. 3.7 – Captura de la Etapa V

### 3.5.7. Etapa VI. Asignación de los materiales

Ésta es otra de las etapas importantes del proceso. Aquí se asignan los materiales a los colores seleccionados anteriormente. Se le muestra al usuario un formulario con los campos correspondientes a los parámetros ($K$, $\rho$, $C_\rho$ y Velocidad) de los materiales para los colores seleccionados anteriormente.

Cuando se introducen los datos, se comprueba si éstos son valores numéricos no nulos y positivos. También se comprueba si se ha dejado vacio alguno de los campos del formulario.

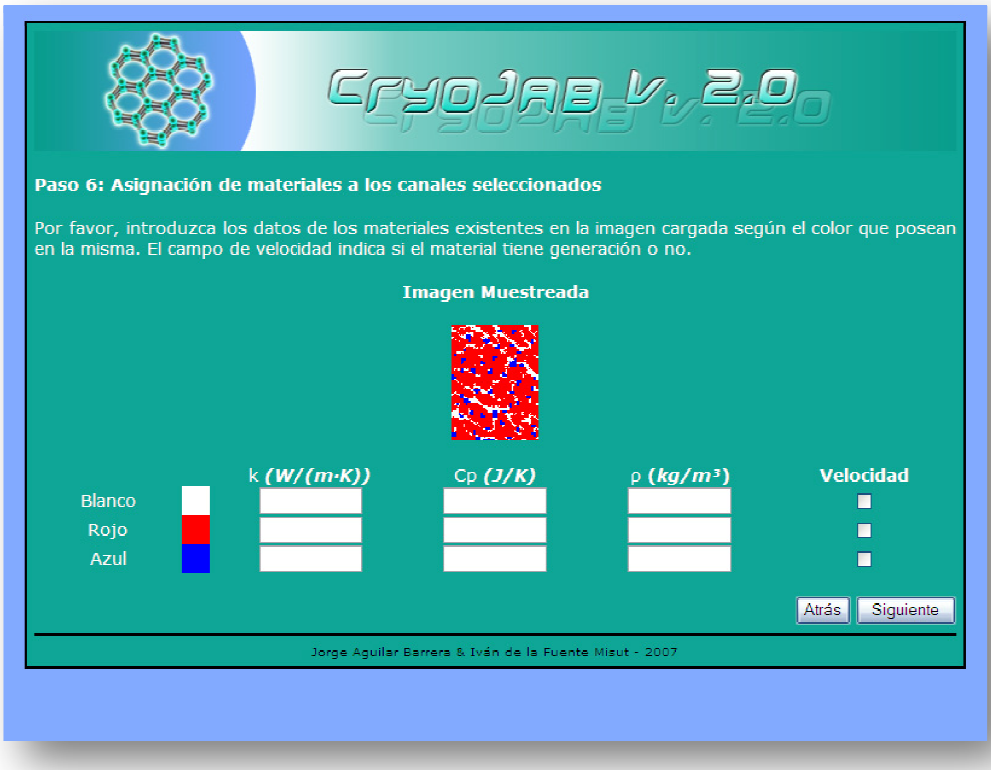



Fig. 3.8 – Captura de la Etapa VI

En todos estos casos se mostraría un error por pantalla, y se liberaría el campo del formulario que ha sido completado de manera errónea, manteniendo los demás. Cuando todos los campos han sido completados se llama a la función que genera el fichero de tipo 2 y se continúa hacia la siguiente etapa.

### 3.5.8. Etapa VII. Introducción de datos de las subestructuras

En este paso el usuario debe rellenar el formulario con los datos de las subestructuras (el tamaño real que posee una subestructura y la temperatura inicial de la misma). El primer parámetro debe haber sido calculado por el usuario. Esto se realiza mediante una división simple del tamaño real de la fotografía (no escalada) entre el número de subestructuras que existen en esa dirección.

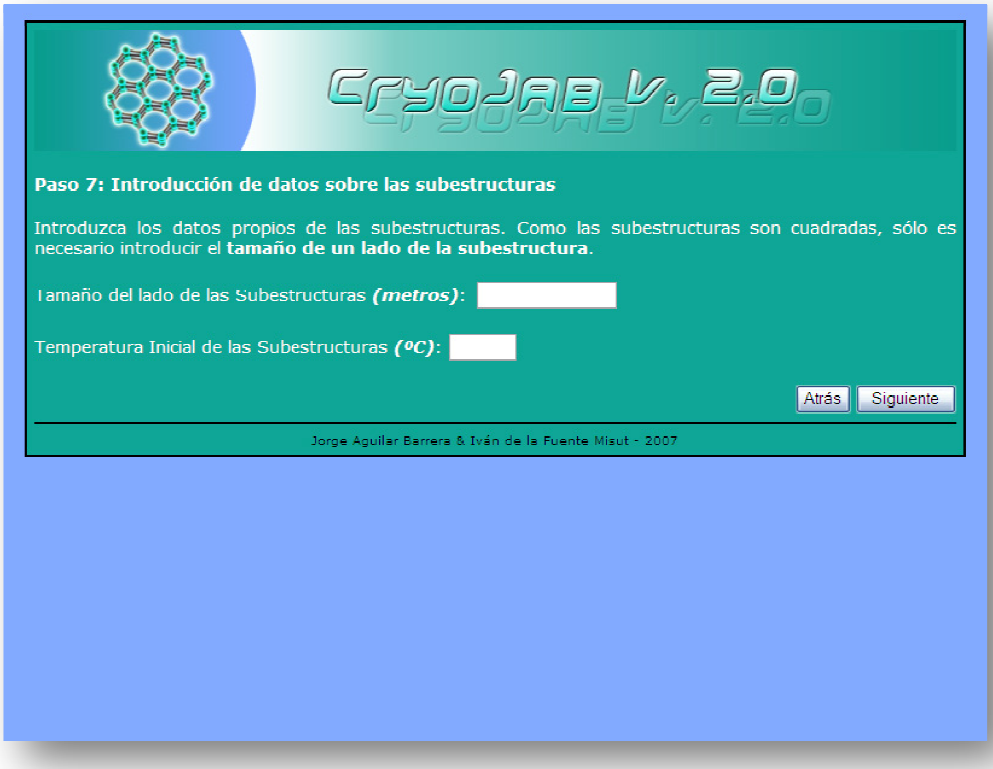



Fig. 3.9 – Captura de la Etapa VII

Sobre los datos introducidos en los campos del formulario se realizan las comprobaciones pertinentes (si el tamaño es un número no nulo y positivo y si no se ha dejado vacio ningún campo).

Una vez completado el formulario se llama a la función encargada de generar los ficheros de tipo 0.

### 3.5.9. Etapa VIII. Introducción del número de condiciones de contorno

En este paso el usuario solo debe indicar el número de condiciones de contorno que existen (de 1 a 4).


Fig. 3.10 – Captura de la Etapa VIII

### 3.5.10. Etapa IX. Introducción de los parámetros de las condiciones de contorno

En este paso se muestra el formulario dinámico para introducir las condiciones de contorno de la fotografía. Aparecerán tantas condiciones como se hayan indicado en el paso anterior.


Fig. 3.11 – Captura de la Etapa IX

El formulario muestra primero la pregunta para cada condición de si ésta es adiabática o no lo es. En caso afirmativo, el formulario se mantiene intacto. En caso negativo se despliegan dos campos más para que el usuario introduzca los parámetros *a* y *b*.

Es importante remarcar que para comodidad del usuario, lo primero que éste debe indicar es el tipo de las condiciones (si son adiabáticas o no), pues si decide rellenar en orden las condiciones, cuando marque el tipo de la siguiente condición, los datos introducidos en los campos de los parámetros de una condición no adiabática se perderían y tendría que volver a rellenar los campos.

Para las condiciones no adiabáticas se comprueba si los datos introducidos en los campos del formulario son coeficientes no nulos. En caso negativo se muestra un error por pantalla.

Si todo está correcto se llama a la función de generación del fichero tipo 4 y se continúa al paso siguiente.

### 3.5.11. Etapa X. Asignación de las condiciones de contorno a la imagen

Esta es la tercera y última etapa relacionada con las condiciones de contorno de la fotografía. Aquí se asignan las condiciones a los lados correspondientes. Por cada lado se muestra una lista desplegable con las condiciones introducidas en el paso anterior (se muestran también las condiciones introducidas por el usuario para que sepa que número se corresponde con cada condición).

Una vez seleccionadas, se llama a la función de generación de ficheros de tipo 3. Esta función calcula automáticamente las subestructuras limítrofes a cada una, asignando los valores correspondientes.

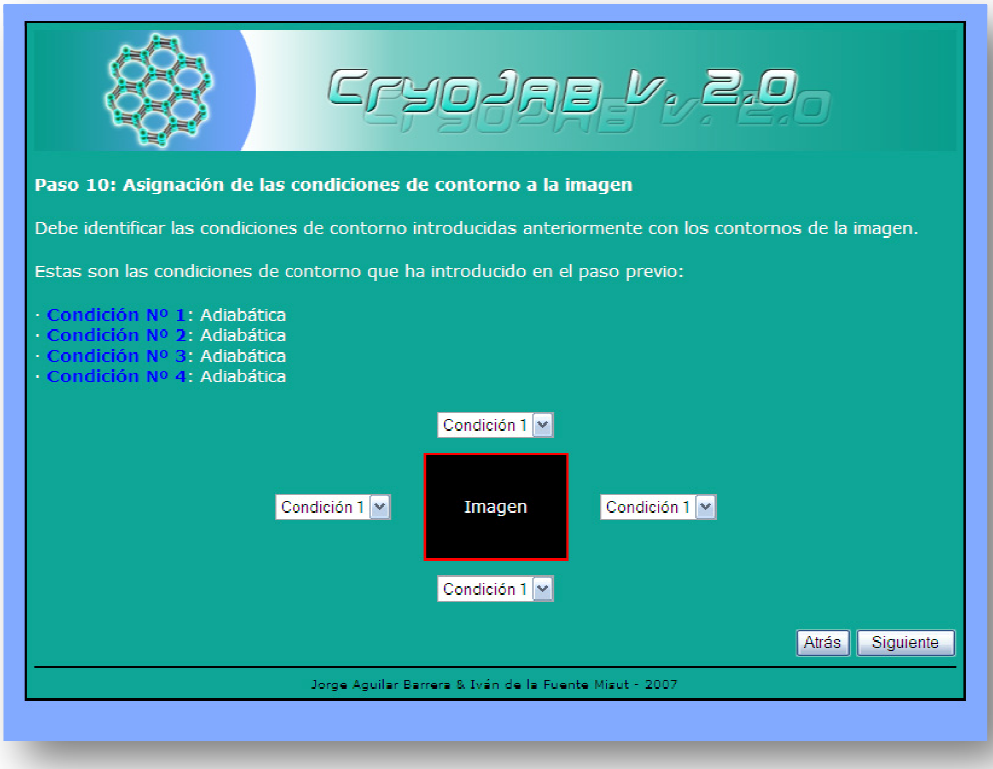



Fig. 3.12 – Captura de la Etapa X

Éste es otro de los grandes pasos que se realiza en el proceso y que libera al usuario de un trabajo tedioso y complejo.

### 3.5.12. 3.5.12. Etapa XI. Introducción de datos de la solución vitrificante

En esta etapa se introducen los datos correspondientes a la solución crioprotectora que se perfusiona a través de los vasos sanguíneos del órgano. Solo se introduce la temperatura inicial del fluido y la velocidad de enfriamiento del mismo. Se comprueba si se han rellenado los dos campos.

Si todo está correcto se llama a la función de generación de ficheros de tipo 5 y se pasa a la etapa siguiente.


Fig. 3.13 – Captura de la Etapa XI

### 3.5.13. Etapa XII. Introducción del tiempo de simulación.

En este último paso de introducción de datos solo se requiere el tamaño del paso de simulación, es decir, el periodo de tiempo que se va a simular. Simplemente se comprueba que el dato introducido es un valor no nulo positivo.

Si todo está correcto se llama a la función de generación de ficheros de tipo 6 y se continúa hacia la siguiente etapa.

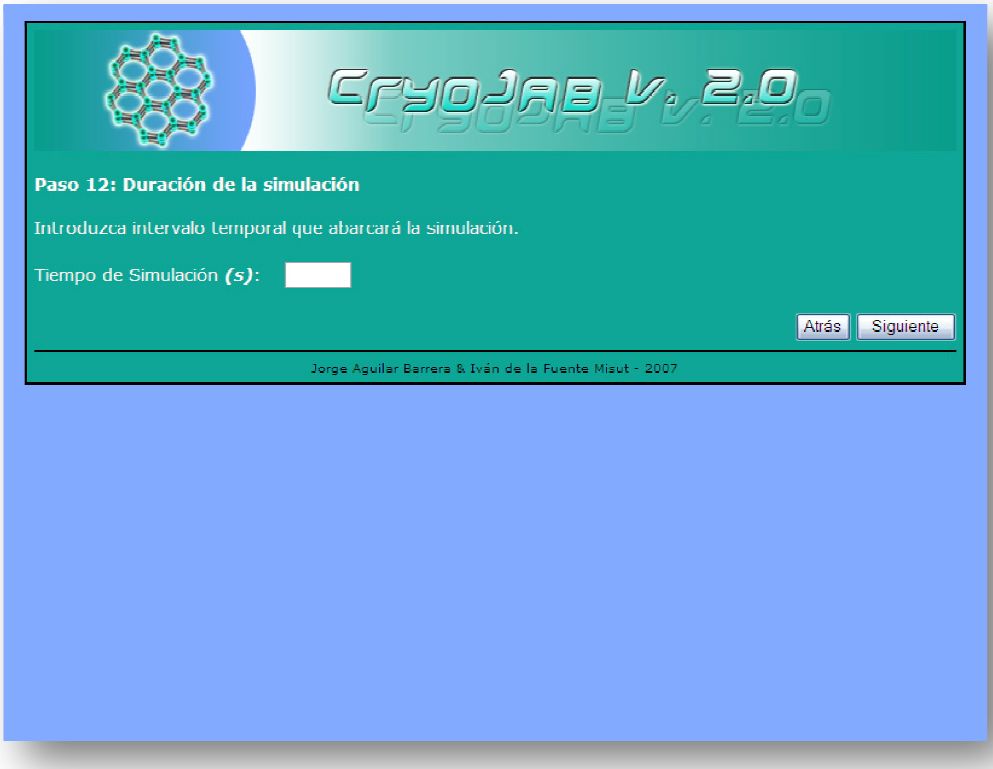



Fig. 3.14 – Captura de la Etapa XII

### 3.5.14. Etapa XIII. Ejecución de la aplicación simuladora.

Ésta es una etapa meramente informativa. Se le comunica al usuario que se va a proceder con la simulación y se le indica el periodo de tiempo que va a simular con ***Cryojab***.


Fig. 3.15 – Captura de la Etapa XIII

### 3.5.15. Etapa XIV. Simulación en curso.

En esta etapa se ejecuta ***Cryojab***. Al usuario se le muestra una pantalla de espera activa, que comprueba periódicamente si ***Cryojab*** ha terminado de simular (se observa si se han generado los ficheros de salida o no).

En el momento en el que los ficheros con los resultados existen, se pasa a la siguiente etapa.


Fig. 3.16 – Captura de la Etapa XIV

### 3.5.16. Etapa XV. Resultados de la simulación (I)

Ésta es la primera etapa de tratamiento de los resultados obtenidos mediante la simulación realizada por ***Cryojab***. Podríamos decir que es la etapa de resultados globales o generales. Se le ofrece la posibilidad al usuario de examinar el fichero con los resultados en forma matricial para todos los instantes de tiempo.

La otra opción es seleccionar un instante concreto para un procesado mayor. Para ello se muestra al usuario una lista desplegable con los instantes de tiempo simulados.


Fig. 3.17 – Captura de la Etapa XV

### 3.5.17. Etapa XVI. Resultados de la simulación (II)

Finalmente se llega a la etapa final de la simulación. Se le ofrece al usuario varias opciones:

- Examinar el fichero con los resultados en forma matricial para el instante de tiempo seleccionado.
- Visualizar gráficamente el resultado para el instante de tiempo seleccionado.
- Volver al inicio de la aplicación para realizar una nueva simulación.
- Cerrar la aplicación.


Fig. 3.18 – Captura de la Etapa XVI

Para llevar a cabo estos pasos lo primero que se ejecuta es una función que extrae los datos en forma matricial del fichero con los resultados globales y lo almacena en un nuevo fichero.

Para la visualización grafica de los resultados se hace uso de ***Matlab***. Es necesario que ***Matlab*** se encuentre instalado en el sistema. No hay que configurar ningún parámetro en ***Matlab*** ni en la aplicación. Simplemente con que se encuentre instalada en el sistema la aplicación hará uso de ella. Para ello genera un fichero de inicio de ***Matlab*** (llamado *startup.m*) en el que se introducen una serie de instrucciones que ***Matlab*** interpreta para generar una grafica acorde a nuestras necesidades y

almacenarla en una imagen. ***Matlab*** se cierra y se muestra en la aplicación la grafica generada en una ventana independiente a la de la aplicación.

Si el usuario decide iniciar una nueva simulación o cerrar la aplicación, todos los ficheros de la simulación generados a lo largo de las diferentes etapas serán eliminados.

## 3.6. Pruebas de funcionamiento del sistema

Para comprobar la correcta implementación de las etapas explicadas en los puntos anteriores, se han llevado a cabo una serie de test que comprueban el buen funcionamiento de los pasos críticos del proceso.

### 3.6.1. Prueba de detección de los colores

El momento crítico que aparece en el proceso es la detección de los colores en la imagen cargada para la simulación. Si esta detección no se realiza de manera correcta, el resultado de la simulación puede ser impredecible y, a todos los efectos, inservible para un posible estudio.

Así que para comprobar que este paso se realiza adecuadamente se ha dispuesto de una serie de pruebas de detección y comprobación mediante la imagen muestreada que genera la función ***Cryojab-color***. Consisten en dos imágenes con contienen los 8 colores que ***Cryojab-color*** es capaz de contemplar en distintas distribuciones y degradaciones, para comprobar si la detección y aproximación es correcta.

La primera prueba consiste en la detección total de todos los colores por aproximación. A la aplicación se le indica que la imagen de entrada contiene todos los colores posibles. La idea es comprobar si detecta correctamente todos los colores.

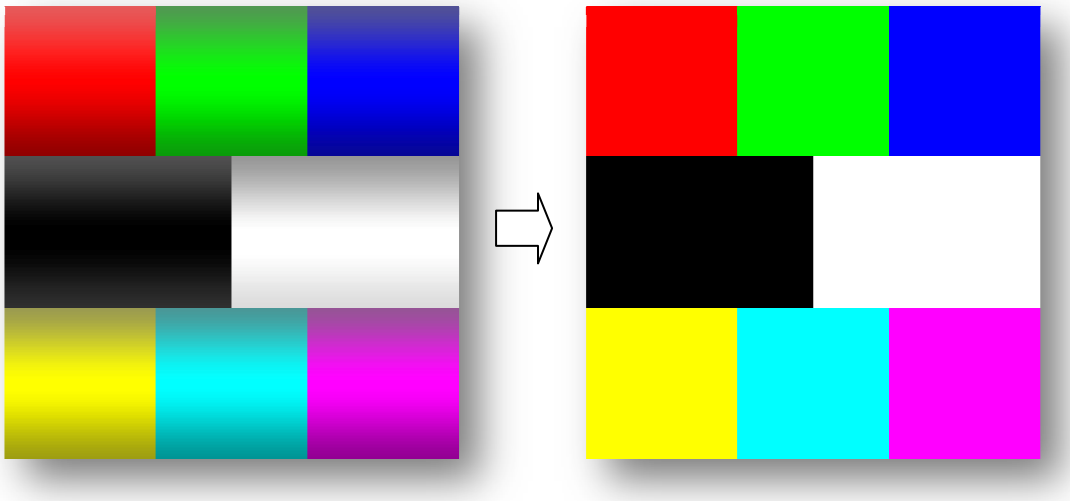


Fig. 3.19 – Imágenes de entrada y salida para comprobación de detección de colores adecuada

Como vemos, la detección se realiza correctamente en ambas imágenes. Los colores degradados son perfectamente detectados.

También hay que comprobar que se detecta adecuadamente el color mayoritario, y que aquellos que no han sido seleccionados son aproximados a éste. Para ello seleccionamos solo el blanco, el azul y el rojo. El blanco es el color mayoritario, así que todo color que no sea alguno de estos tres debe ser aproximado al blanco.

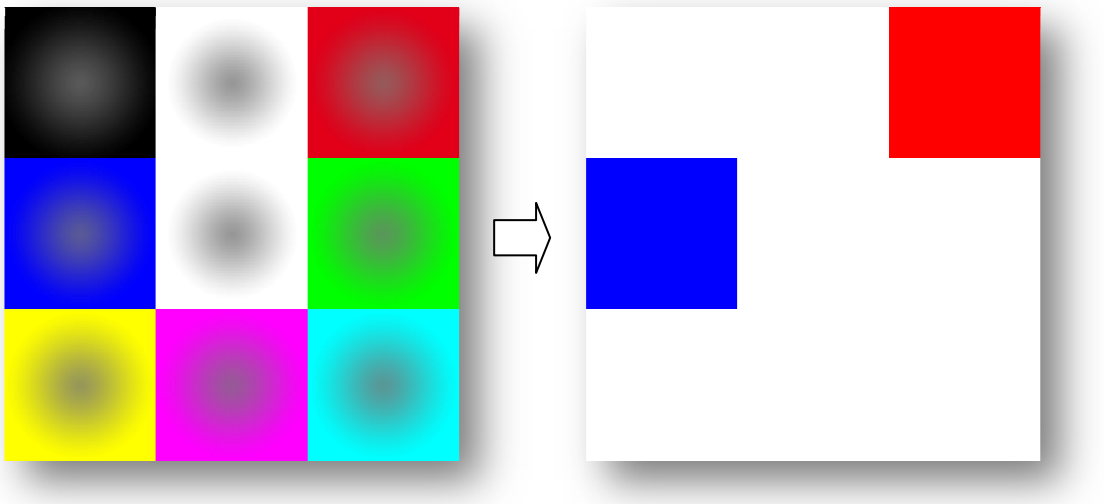


Fig. 3.20 – Imágenes de entrada y salida para comprobación de aproximación de colores adecuada

Como vemos, las dos pruebas de detección de colores son satisfactorias. Así que podemos concluir que la aplicación de detección de colores funciona de manera adecuada.

### 3.6.2. Prueba de generación

En este test se comprueba si la generación se realiza de manera adecuada. Es decir, que si se detecta correctamente que material es un capilar (en el que existe generación), y cómo evoluciona la temperatura a lo largo del tiempo. Para ello utilizaremos una imagen con una zona que consideraremos capilar y otra que no, y veremos si los resultados son coherentes. Se usan los mismos parámetros de simulación para los dos materiales. La única diferencia es que el azul es capilar y el rojo no lo es.

Se consideran todos los lados de la imagen adiabáticos. El fluido enfría a -1ºC/s. Y tanto el fluido como los materiales se encuentran inicialmente a 36ºC.

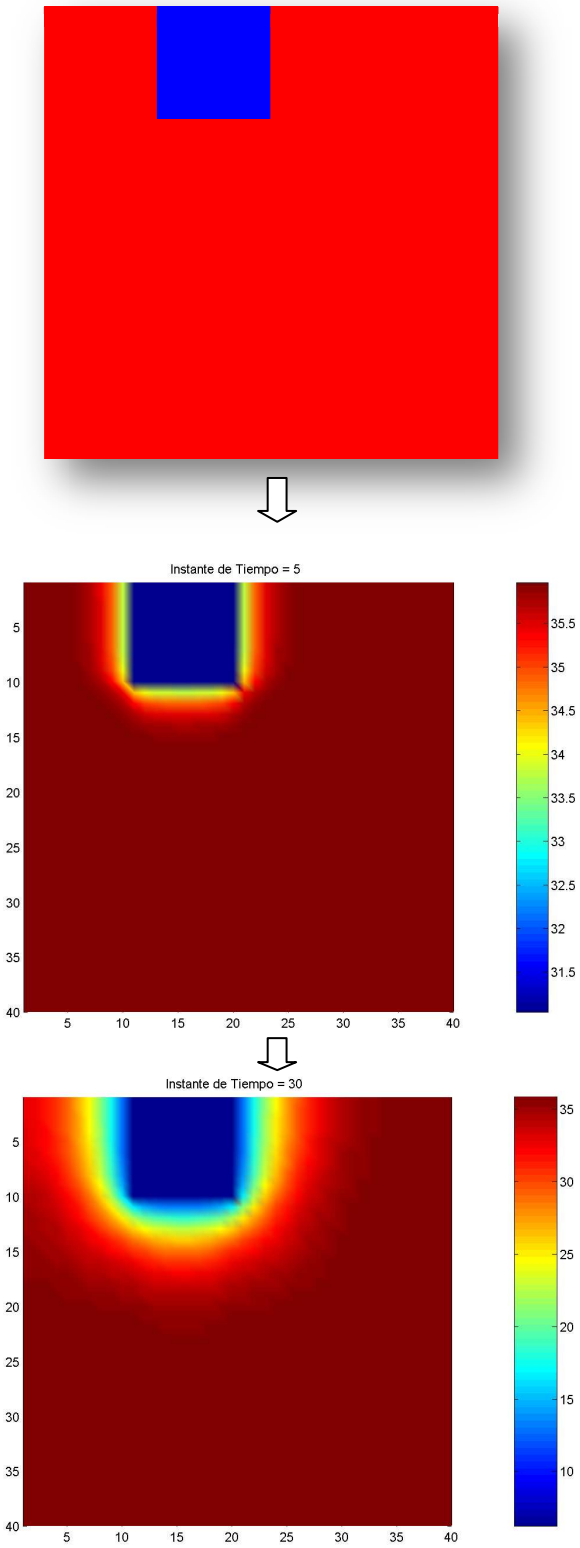


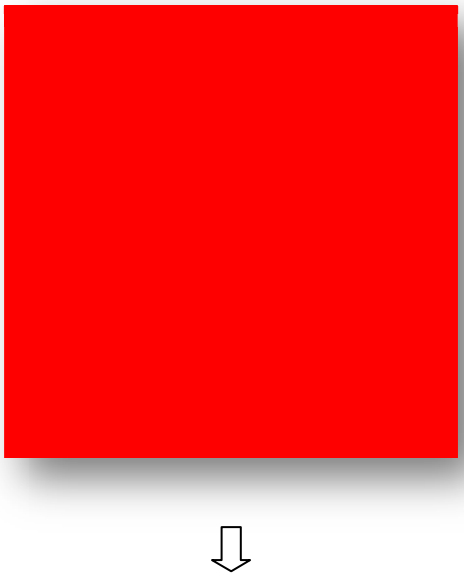

Fig. 3.21 – Imágenes de entrada y salida (5 y 30 segundos) para comprobación de generación adecuada

Como vemos, la generación se realiza de manera adecuada, transmitiéndose la temperatura del capilar al resto de los materiales de forma progresiva a lo largo del tiempo.

### 3.6.3. Prueba de errores de nulidad

En este test se quiere comprobar si la aplicación devuelve resultados coherentes si fijamos la temperatura de dos de los lados de la fotografía.

Para ello se usa una imagen homogénea, con un único material sin generación (que no es un capilar), y se fijan las paredes laterales a una determinada temperatura. Concretamente a 15ºC el lado izquierdo y a 5ºC el lado derecho. Las paredes superior e inferior se consideran adiabáticas. Se usan los mismos parámetros de simulación que en la prueba anterior.



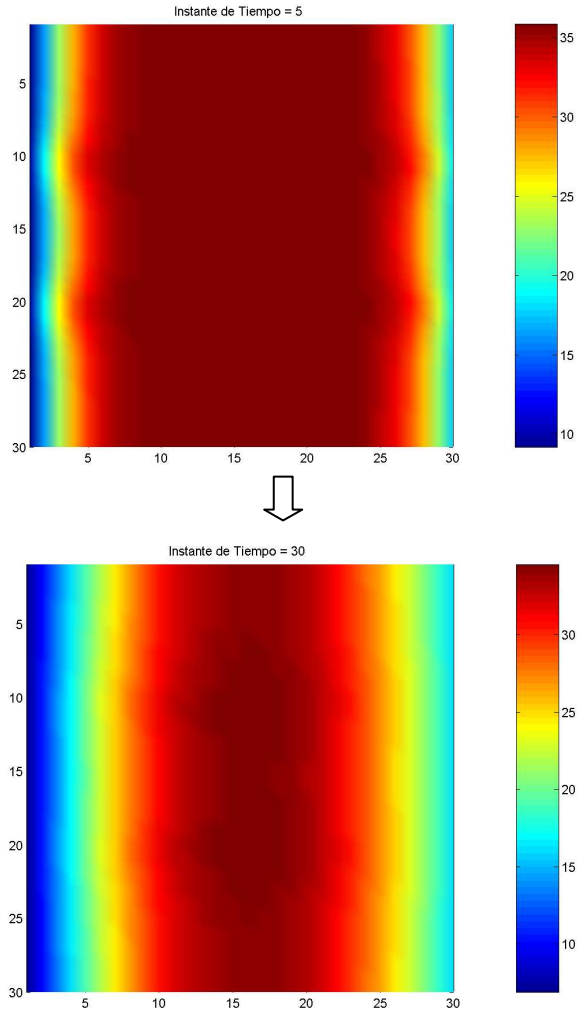



Fig. 3.22 – Imágenes de entrada y salida (5 y 30 segundos) para comprobación de errores de nulidad

Como vemos, las salidas son coherentes con lo que esperamos. La temperatura se transmite a través de las paredes laterales de manera progresiva, con mayor intensidad en la izquierda que en la derecha.

### 3.6.4. Conclusiones de las pruebas de funcionamiento

Como hemos podido ver, el sistema funciona correctamente. Realiza una detección muy fiel de los colores seleccionados y crea los ficheros necesarios para que ***Cryojab*** lleve a cabo la simulación de forma correcta.

Así pues, ya tenemos a punto el sistema para poder realizar un estudio exhaustivo para diferentes órganos en distintas condiciones con mucha más velocidad y fiabilidad.

## 3.7. Diagramas de flujo

En este apartado se va a ofrecer los diagramas de flujo de los procesos que se llevan a cabo a lo largo de toda una simulación, excluyendo el correspondiente a la aplicación original Cryojab, que ya se expuso anteriormente.

Antes que nada resaltar que al tratarse de una aplicación web, en cada paso se realizan una serie de comprobaciones que son prácticamente idénticas. Entonces, en lugar de únicamente adjuntar un diagrama de flujo por cada paso que se realice, se va a adjuntar un diagrama de flujo por cada pequeño subproceso que se realice, independientemente del paso en el que se lleve a cabo. Posteriormente, se adjuntara un diagrama de flujo con el proceso completo, es decir, la realización completa de una simulación de principio a fin.

### 3.7.1. Diagrama de flujo de la función *limpieza*

Esta función se encarga de vaciar el directorio donde se encuentra la aplicación principal (***Cryojab*** e ***ImageMagick***). Comprueba cada fichero si es una aplicación ejecutable o no. En caso negativo se elimina. El objetivo de esta función no es otro que asegurar que los datos de una determinada simulación no van a interferir en la siguiente.

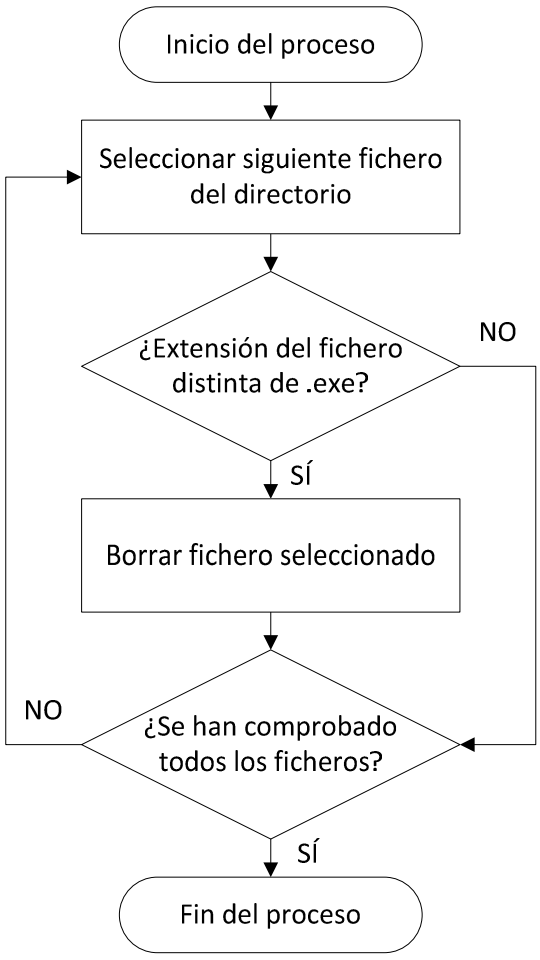



Fig. 3.23 – Diagrama de flujo de la función *limpieza*

### 3.7.2. Diagrama de flujo de comprobación de errores en datos del formulario

A continuación se muestra el diagrama de flujo del proceso básico que se lleva para comprobar datos erróneos introducidos mediante los formularios. Existen una serie de errores a comprobar en función de los datos introducidos (imagen correcta, datos numéricos, datos positivos, datos distintos de cero…), pero el proceso que se sigue es el mismo para todos, así que usaremos un diagrama de flujo genérico.

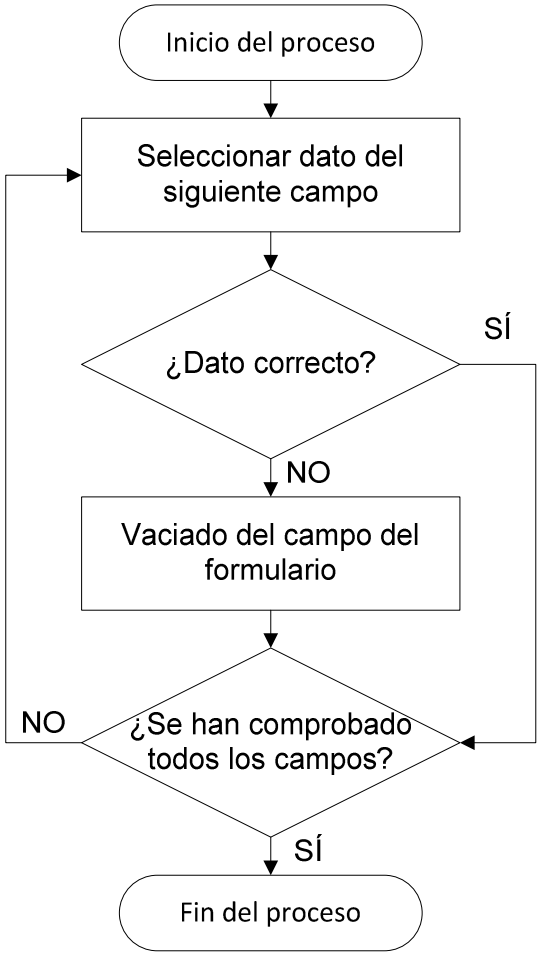



Fig. 3.24 – Diagrama de flujo de comprobación de errores en datos de formulario

### 3.7.3. Diagrama de flujo de la función *fichero0*

Esta función es la encargada de generar los ficheros de tipo 0 necesarios para la simulación. Recibe los siguientes parámetros: el número de nodos en horizontal o vertical en una subestructura, el número de subestructuras en horizontal y en vertical, el tamaño de la subestructura en horizontal y en vertical y la temperatura inicial de las subestructuras. La función calcula el número total de subestructuras y mediante un bucle genera un fichero idéntico con los datos necesarios por cada subestructura.

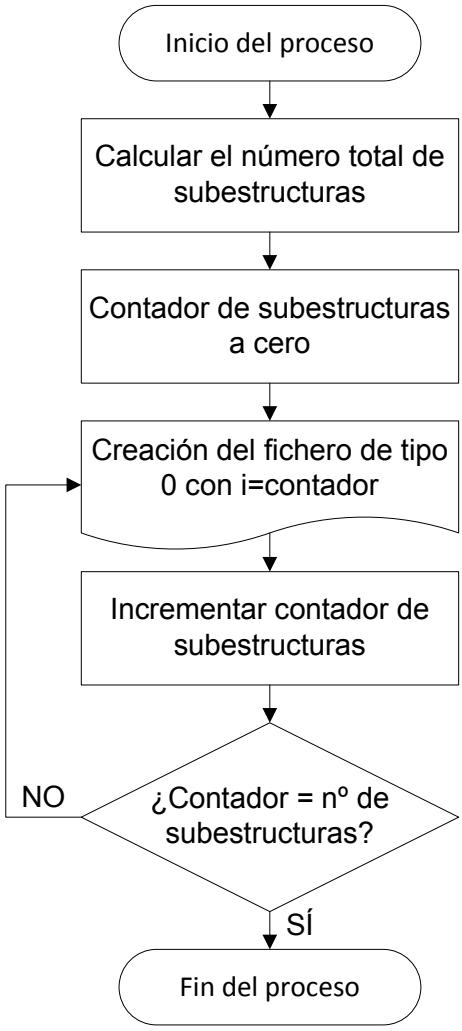



Fig. 3.25 – Diagrama de flujo de la función *fichero0*

### 3.7.4. Diagrama de flujo de la función *fichero1*

Esta función se encarga de generar los ficheros de tipo 1 necesarios para la simulación. Recibe los siguientes parámetros: un array con los colores seleccionados, el número de nodos en horizontal o vertical y la resolución de la imagen. Esta función se encarga de generar el fichero de entrada que necesita la función *cryojab_color*, realiza la llamada a *cryojab_color* y convierte a formato **PNG** la imagen textual que genera *cryojab_color*.

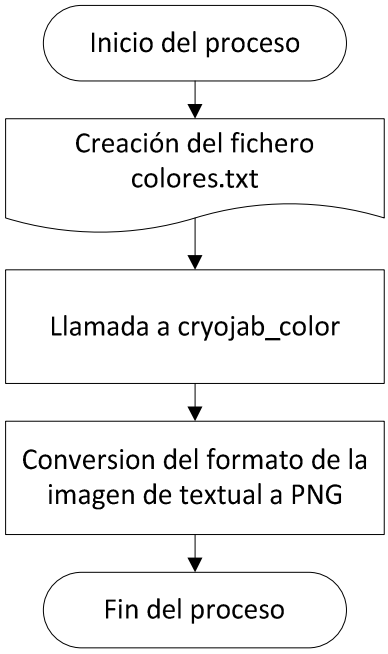



Fig. 3.26 – Diagrama de flujo de la función *fichero1*

### 3.7.5. Diagrama de flujo de la función *cryojab_color*

Ésta es la función más compleja de toda la aplicación. Es la encargada de realizar la detección de los colores en la imagen. Recibe como parámetros el número de

nodos en vertical o en horizontal de una subestructura, así como el número de subestructuras en horizontal y en vertical. Además requiere de la existencia del fichero *colores.txt*. El proceso, como se vio anteriormente, consiste en una prelectura del fichero con la imagen para detectar el color mayoritario, luego una lectura con asignación de los colores a cada pixel y, finalmente, un volcado a los ficheros de tipo 1 y al fichero con la imagen muestreada.

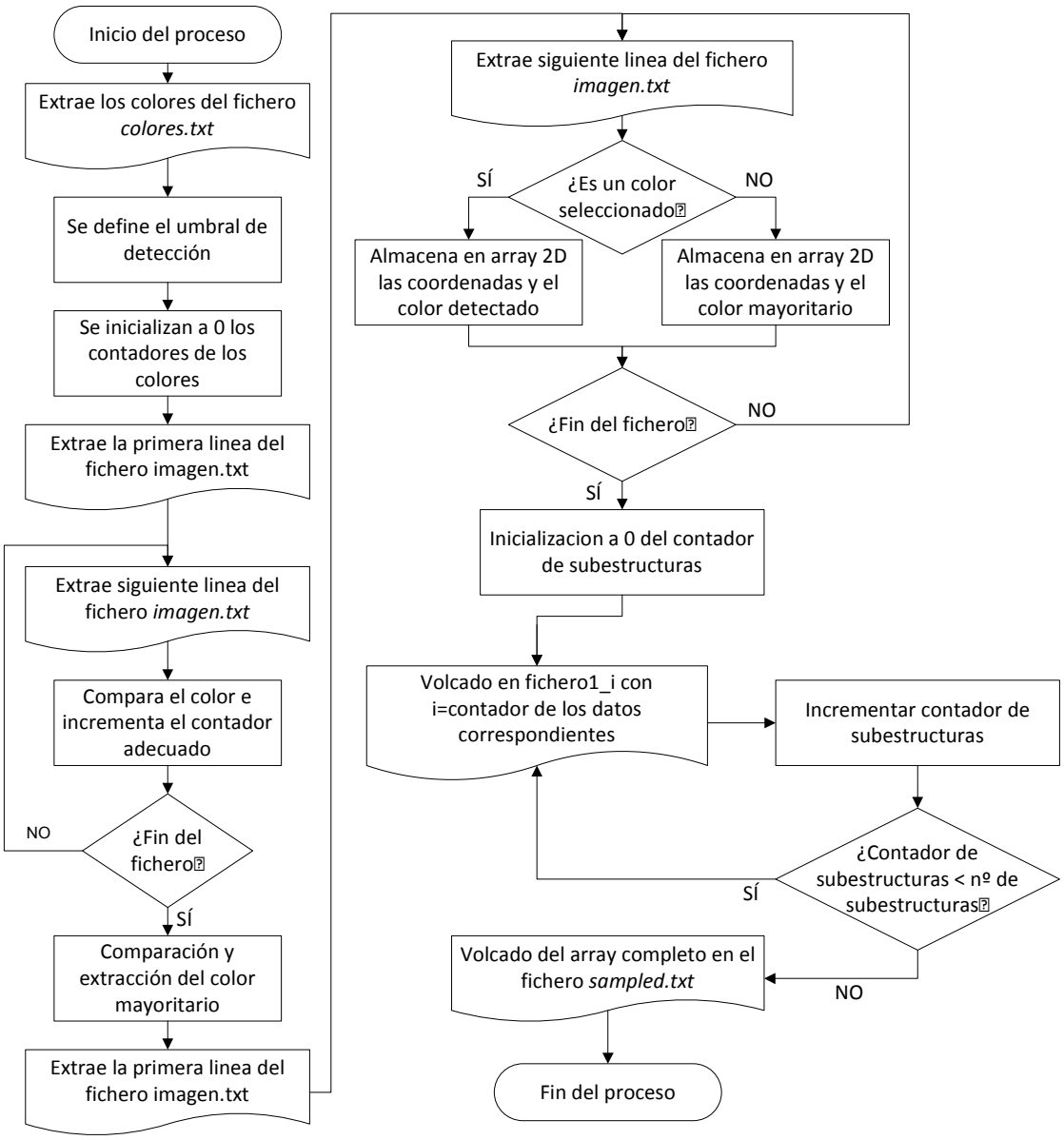



Fig. 3.27 – Diagrama de flujo de la función *cryojab_color*

## 3.7.6. Diagrama de flujo de la función *fichero2*

Ésta función se encarga de generar el fichero de tipo 2 necesario para la simulación. Recibe los siguientes parámetros: recibe el array completo del formulario y el número de materiales. Como el array posee los datos almacenados de manera ordenada, se vuelcan directamente en el fichero. Solo hay que asegurarse de que no se almacenan los datos innecesarios.

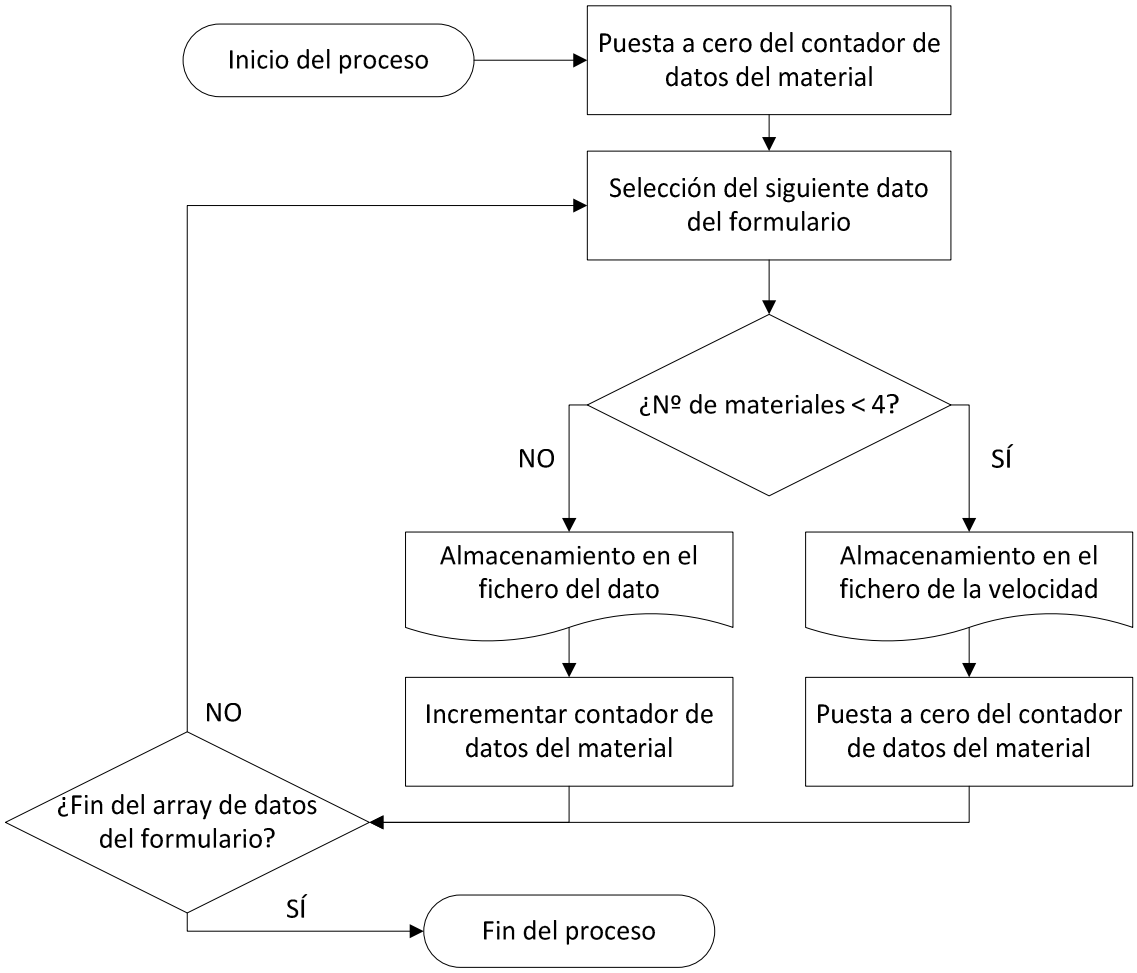



Fig. 3.28 – Diagrama de flujo de la función *fichero2*

### 3.7.7. Diagrama de flujo de la función *fichero3*

Ésta es la función encargada de generar los ficheros de tipo 3 necesarios para simulación. Recibe los siguientes parámetros: el número de subestructuras en horizontal y en vertical, así como las condiciones de contorno de la fotografía. Para cada subestructura se calculan las subestructuras limítrofes.

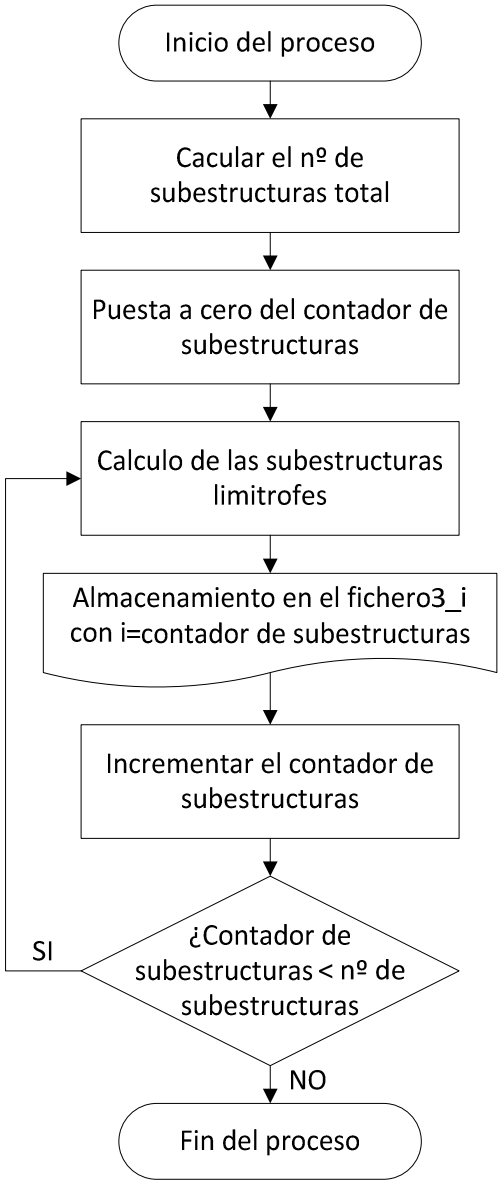



Fig. 3.29 – Diagrama de flujo de la función *fichero3*

## 3.7.8. Diagrama de flujo de la función *fichero4*

Ésta es la función que se encarga de generar el fichero de tipo 4 necesario para la simulación. Recibe los siguientes parámetros: el array de los datos generados por el formulario de la página y el número de condiciones de contorno distintas.

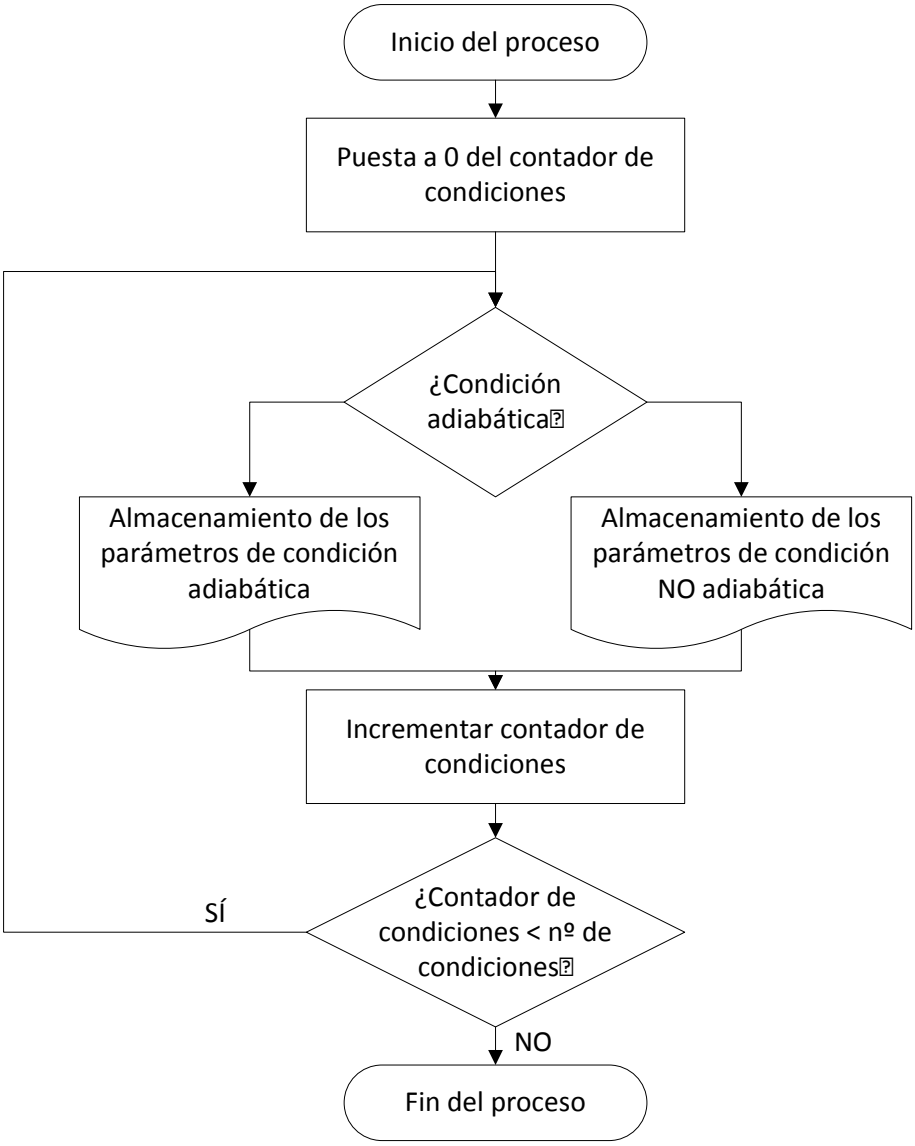



Fig. 3.30 – Diagrama de flujo de la función *fichero4*

### 3.7.9. Diagrama de flujo de la función *fichero5*

Esta es la función encargada de generar el fichero de tipo 5 necesario para la simulación. Recibe los siguientes parámetros: el número de subestructuras, la temperatura inicial del fluido y la velocidad del fluido. Simplemente se almacena una línea con los datos del fluido por cada subestructura.

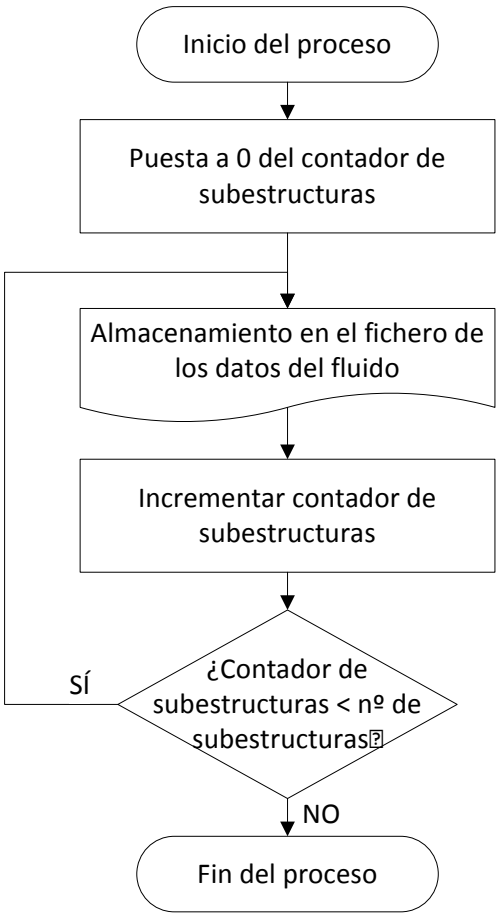



Fig. 3.31 – Diagrama de flujo de la función *fichero5*

### 3.7.10. Diagrama de flujo de la función *fichero6*

Esta función se encarga de generar el fichero de tipo 6 necesario para realizar la simulación. Recibe los siguientes parámetros: el número de subestructuras en horizontal y en vertical y el tiempo de simulación. Simplemente se vuelcan los datos en el fichero.

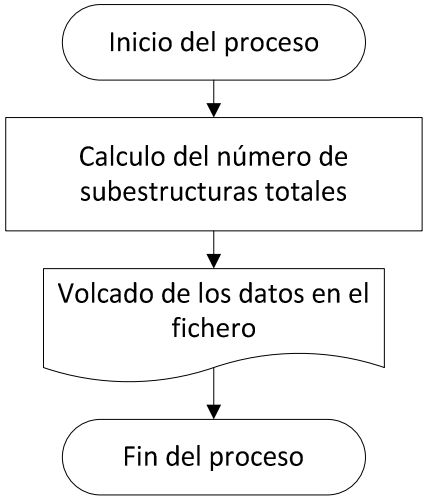



Fig. 3.32 – Diagrama de flujo de la función *fichero6*

### 3.7.11. Diagrama de flujo de la función *ext_res*

Esta función se encarga de extraer los datos pertenecientes al instante de tiempo solicitado del fichero con todos los datos de la simulación realizada. Recibe como único parámetro el instante de tiempo a extraer. Simplemente se realiza una búsqueda en el fichero completo hasta que se encuentra el instante indicado. Entonces se copia el contenido del fichero a uno nuevo hasta que se encuentre el instante siguiente.

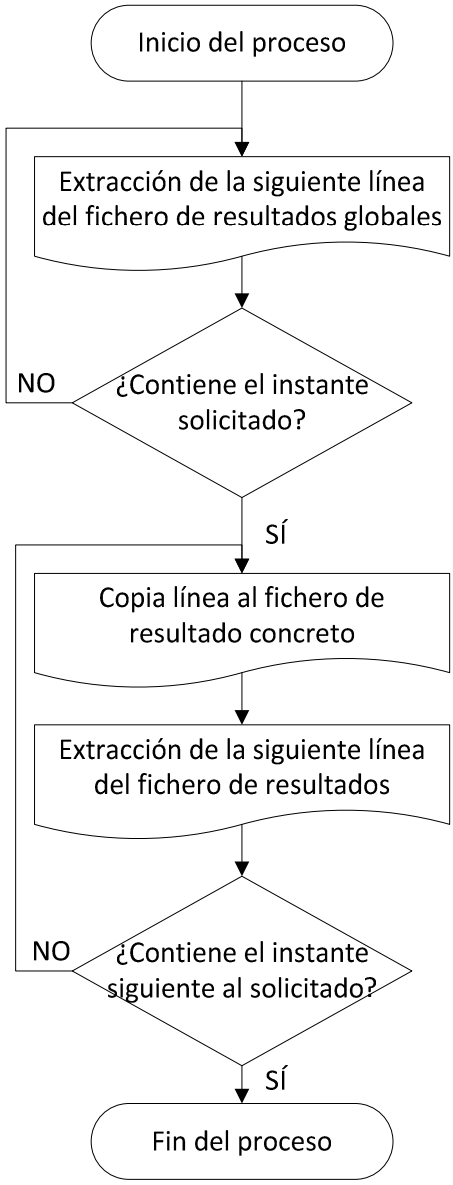



Fig. 3.33 – Diagrama de flujo de la función *ext_res*

### 3.7.12. Diagrama de flujo de la función *call_matlab*

Esta función se encarga de generar el fichero *startup.m* necesario para realizar la llamada a ***Matlab*** y poder obtener la visualización grafica de los datos de un instante de tiempo concreto. Posteriormente realiza una llamada a la aplicación ***Matlab***.

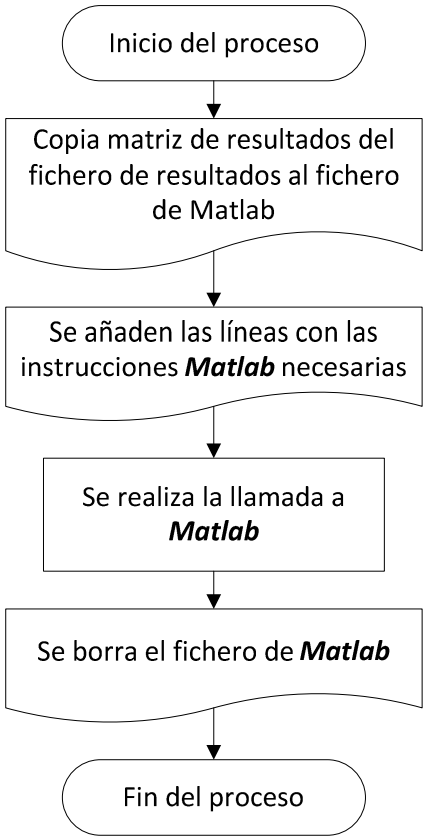



Fig. 3.34 – Diagrama de flujo de la función *call_matlab*

### 3.7.13. Diagrama de flujo del proceso completo de una simulación

Ya tenemos todos los diagramas de flujo de las funciones y procesos secundarios para desarrollar el diagrama de flujo del proceso completo. Es importante resaltar que este proceso no es un proceso automático, sino que necesita de la colaboración del usuario, y en cada carga de etapa existe la posibilidad de volver a la etapa anterior deshaciendo los cambios (esto no se ha reflejado en el diagrama, pues aporta muy poca información y complica el diagrama en demasía). El paso de una etapa a otra viene provocado porque el usuario ha decidido continuar adelante. Así que suponemos que se realiza el proceso completo sin que el usuario vuelva atrás en ningún momento.

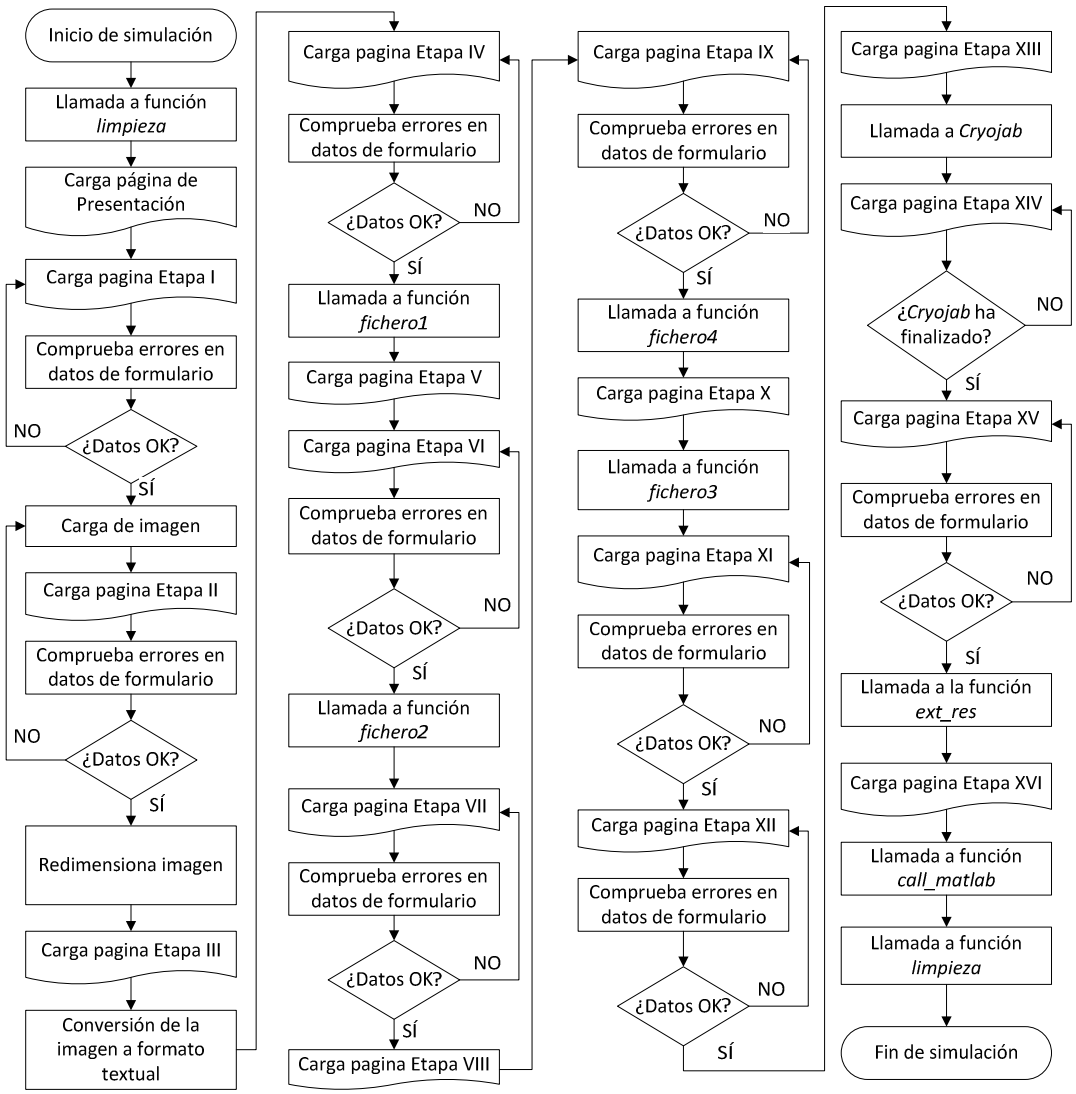



Fig. 3.35 – Diagrama de flujo de la realización de una simulación