4 SOLUCIÓN DEFINITIVA

4.1 INTRODUCCIÓN

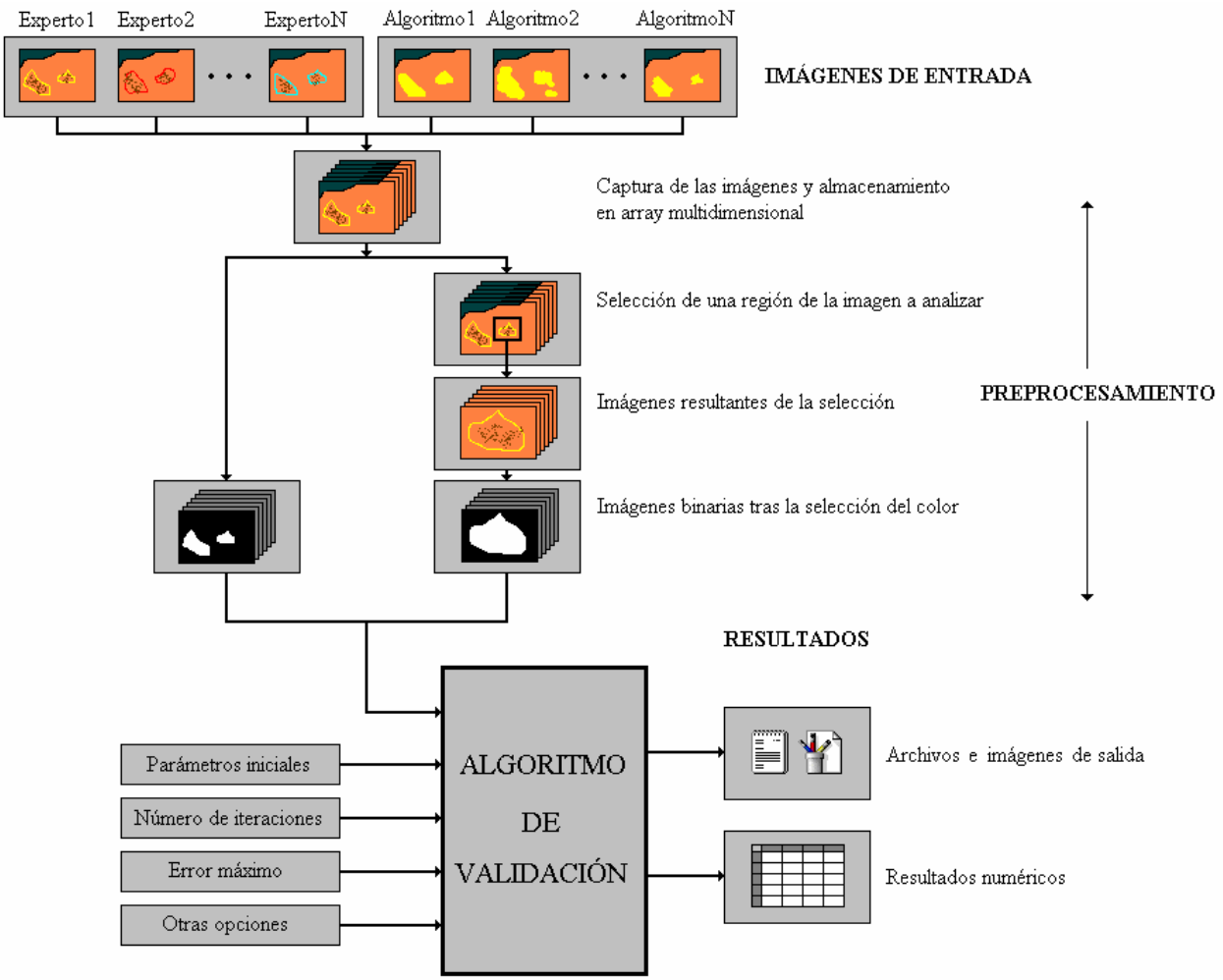
En este capítulo se analizan detenidamente todos los aspectos relativos a la solución final encontrada. Dicha solución se basa, obviamente, en el algoritmo de validación de los procesos automáticos de segmentación de imágenes.

Sin embargo, hay que destacar que, a pesar de ser necesaria, ésta no es suficiente para dar por concluido el proyecto, dado que supone únicamente lo que podría llamarse el núcleo del programa.

Además del algoritmo, es preciso diseñar una serie de programas que permitan modificar las imágenes de entrada. Éstas tienen un gran tamaño, se trata de imágenes a color (con varios contornos correspondientes a las decisiones de los expertos también en colores diferentes), e incluso pueden no tener las mismas características. Tras el debido preprocesamiento, se facilitará al núcleo del programa (algoritmo de validación) un array multidimensional que contendrá imágenes binarias y de idénticas propiedades.

Por último, dada la gran cantidad de variables que hay que utilizar para configurar el funcionamiento del programa, así como la existencia de un abundante número de resultados, ya sean numéricos o gráficos, se ha considerado conveniente el diseño de una interfaz gráfica en MATLAB gracias a la cual la interacción entre el usuario y el programa sea lo más cómoda posible.

Para aportar una idea general del funcionamiento del programa definitivo, se adjunta a continuación un diagrama en el que se incluyen los principales pasos a seguir durante la ejecución:



4.2 EL PROCESAMIENTO DE LAS IMÁGENES

4.2.1 INTRODUCCIÓN

Como ya se ha comentado, el diseño del programa no consiste únicamente en hallar un algoritmo de validación correcto. Una vez que éste es encontrado, su traducción a un programa en MATLAB es una labor relativamente sencilla. No ocurre lo mismo a la hora de capturar las imágenes y procesarlas, ni cuando se pretende mostrar los resultados o permitir diversas opciones en cuanto a la ejecución del programa.

En los apartados sucesivos se explican varias operaciones indispensables para un correcto funcionamiento.

4.2.2 CAPTURA Y ALMACENAMIENTO DE LAS IMÁGENES EN MEMORIA

Tal y como se ha explicado en los comentarios de algunos programas ya tratados anteriormente, las imágenes de entrada se almacenan en un array multidimensional llamado normalmente 'A' o 'B'.

Cuando las imágenes son binarias, se requiere un array de dimensión 3, para indicar fila, columna, e indexar la imagen en cuestión, es decir, A(i,j,k) indicará el valor del píxel de coordenadas (i,j) relativo a la imagen k-ésima (que puede ser la decisión de un experto o el resultado de la segmentación de un algoritmo sujeto a examen).

Sin embargo, cuando se trabaja con imágenes a color, la dimensión del array debe ser una unidad mayor que en el caso anterior, dado que hay que especificar las componentes RGB del píxel. Por tanto, se escribirá A(i,j,rgb,k) para hacer referencia al valor de las componentes R (A(i,j,1,k), G (A(i,j,2,k)) y B (A(i,j,3,k)) del píxel (i,j) de la imagen k-ésima.

Esta forma de almacenamiento es bastante acertada dado el desconocimiento del número de imágenes de entrada y la necesidad de utilizar expresiones genéricas para efectuar determinados cálculos (tan sólo con ir modificando un índice, podemos pasar de una imagen a otra).

En lo referente a la captura inicial de las imágenes hay que tener en cuenta que, en un principio, las imágenes procedentes de los expertos se hallaban agrupadas en archivos de PowerPoint. Por tanto, se han ido copiando una a una a archivos individuales en formato BMP. Las imágenes proporcionadas por los programas de segmentación están contenidas también en archivos de tipo BMP, aunque se pueden encontrar bajo formato JPG.

En ambos casos, puede utilizarse la orden *imread* de MATLAB para capturar dichas imágenes.

Ejemplo:

```
A(:,:,:,k) = imread(varargin{k});
```

Este mecanismo de almacenamiento en arrays multidimensionales permite capturar todas las imágenes que se deseen con tan sólo escribir la línea anterior dentro de un bucle de tipo *for* que haga variar k entre 1 y *nargin* (número de argumentos a la entrada).

4.2.3 PREPROCESAMIENTO DE LOS RESULTADOS DE LA SEGMENTACIÓN

Las imágenes correspondientes a los resultados de los programas de segmentación a validar no se consideraron, en principio, problemáticas.

No obstante, cuando comienzan a ejecutarse los primeros programas, se producen fallos que los interrumpen. Esto da lugar cuando dichas imágenes vienen incluidas en formato JPG, y consiste en una discrepancia respecto de los tamaños de las imágenes correspondientes a los expertos (es una condición obligatoria que sean idénticos para poder proceder con la forma de almacenamiento explicada en 4.2.2).

El motivo que hacía diferir los tamaños era un grueso marco blanco que rodeaba las primeras imágenes (JPG), resultante, posiblemente, de copiar y pegar cuando se visualizaron en su momento los resultados del algoritmo de segmentación. Por tanto, se hizo necesario idear un mecanismo para eliminar dicho marco residual y reajustar también el tamaño de la propia imagen, que cambiaba ligeramente (***prepara_alg.m***).

El programa en MATLAB es el siguiente:

-- PREPARA_ALG.M --

```
% FUNCIÓN QUE ADAPTA LA IMAGEN 'A' AL TAMAÑO TAM,
% ELIMINANDO EL MARCO BLANCO QUE LA RODEA, SI ES QUE EXISTE.

% alto: size(I,1)
% ancho: size(I,2)

function B = prepara_alg (A,tam)


i=1;
j=1;
flag=1;
x1=1;
y1=1;
x2=1;
y2=1;
margen=100;
```

```
% Detección de la esquina superior izquierda de la imagen
% rodeada por el marco blanco.

for i=1:size(A,1)
   for j=1:size(A,2)

        % Se tendrá en cuenta que la posición actual no pertenece al marco blanco
        % si el valor del píxel difiere de dicho color (255) en un margen mínimo
        % considerable.

        if ((abs(double(A(i,j,1))-255)>=margen)|(abs(double(A(i,j,2))-
                    255)>=margen)|(abs(double(A(i,j,3))-255)>=margen))&flag

            % Coordenadas esquina:
            x1=j;
            y1=i;

            % Fin del bucle:
            j=size(A,2);
            i=size(A,1);
            flag=0;
        end
    end
end

flag=1;
% Detección de la esquina inferior derecha de la imagen rodeada por el marco
% blanco.

for i=size(A,1):-1:1
   for j=size(A,2):-1:1
        % Se tendrá en cuenta que la posición actual no pertenece al
        % marco blanco si el valor del píxel difiere de dicho color
        % (255) en un margen mínimo considerable.

        if ((abs(double(A(i,j,1))-255)>=margen)|(abs(double(A(i,j,2))-
                    255)>=margen)|(abs(double(A(i,j,3))-255)>=margen))&flag

            % Coordenadas esquina:
            x2=j;
            y2=i;
            % Fin del bucle:
            j=1;
            i=1;
            flag=0;
        end
    end
end

% Dadas las coordenadas de las esquinas, se elimina el marco y se
% captura en 'B' la imagen en cuestión:

B = imcrop(A,[x1 y1 x2-x1 y2-y1]);

% Se reajusta el tamaño para hacerlo coincidir con el de las imágenes
% de los expertos, cuyo tamaño viene dado en el vactor 'tam':

B = imresize(B,[tam(1,1),tam(1,2)]);

return
```

IMPORTANTE:

Nada de esto último es necesario si se modifica convenientemente el programa de segmentación. Mediante el comando *imwrite* de MATLAB, es posible guardar directamente el resultado de la segmentación en un fichero de cualquier tipo (por ejemplo, BMP), de tal forma que no se requiere procesado sobre el mismo (la imagen resultante es de igual tamaño que la original, y no aparecen alteraciones de ningún tipo).

4.2.4 EL PROBLEMA DEL TIEMPO DE PROCESO

Como ya se comentó en el capítulo anterior, las imágenes poseen un tamaño considerable y es preciso recorrerlas píxel por píxel en innumerables ocasiones (para la detección del color, durante las múltiples iteraciones del algoritmo de validación, etc.). Por tanto, convendría encontrar una solución al problema de los enormes tiempos de ejecución de los programas.

Podría ser una alternativa viable, para imágenes en las que la quemadura comprende un contorno pequeño, la utilización de la instrucción *imcrop* para crear una imagen correspondiente a la subregión que contenía dicho contorno, ya que así se trabajaría con imágenes que ocupan mucha menos memoria.

No obstante, se ha optado por ejecutar el programa con las imágenes en su total extensión, asumiendo el inconveniente citado.

4.2.4.1 Inconvenientes de la solución

Uno de los principales inconvenientes de esta solución es que se requiere un cuadro de selección idéntico para todas las imágenes, tanto en tamaño (para no tener problemas de almacenamiento) como en las coordenadas de la selección (para que así el algoritmo de validación no interprete desplazamientos inexistentes entre los contornos de diferentes imágenes). Una vez estudiada la orden *imcrop* en detalle, se confirma la posibilidad de poder almacenar las coordenadas del primer cuadro de selección (que será el único creado manualmente) y ejecutar la instrucción automáticamente y con las mismas coordenadas en el resto de imágenes, resolviéndose así el problema (esta operación se halla comentada en el código del programa definitivo).

El otro inconveniente que se presenta se basa en un aspecto relativo al algoritmo de validación elegido finalmente. Consiste en que la elección de una subregión determinada repercute en los resultados finales del algoritmo de validación, es decir, para selecciones diferentes se pueden obtener resultados también diferentes, si no se tiene en cuenta cierto parámetro de entrada (que se llamará g(T)) y debe ser la porción del área total de la imagen que ocupa la región significativa.

4.2.5 LA DETECCIÓN DEL COLOR

Una vez que se dispone de la imagen original o de una imagen menor (región marcada en la anterior), es necesario obtener, a partir de ella, otra imagen que sea binaria, resultante de la elección del color con que cada experto ha seleccionado, según su opinión, la región significativa. Los píxeles de la nueva imagen valdrán 1 si forman parte del contorno seleccionado con el color considerado (o de su interior), mientras que el resto valdrá 0.

La base de este programa es la instrucción *impixel* de MATLAB, que devuelve las componentes RGB exactas de un píxel seleccionado con el ratón sobre la imagen.

Es recomendable, para imágenes relativamente grandes o contornos dibujados con pinceles estrechos, realizar una operación *imcrop* que se utilice como zoom para así asegurar al usuario pinchar exactamente en un píxel del color buscado.

Tras estas aclaraciones, se muestra el correspondiente programa de detección del color:

-- DETECTA_COLOR.M --

```
% Funcion que devuelve una imagen binaria resultante de la deteccion de un
% determinado color en la imagen de entrada M:
function A = detecta_color(M,c,margen)

i=1;
j=1;

for i=1:size(M,1)
   for j=1:size(M,2)
       % Se comprueba que el color del píxel (i,j) de la imagen es,
       % aproximadamente, el especificado:

       if proximo(M,c,margen,i,j,1) & proximo(M,c,margen,i,j,2) &
                                                        proximo(M,c,margen,i,j,3)
          A(i,j)=1;
       else
          A(i,j)=0;
       end
   end
end
return
```

-- PROXIMO.M --

```
% Subprograma que devuelve 1 si el vector 1x3 correspondiente al color del píxel
% (i,j) de la imagen k-ésima contenida en el array multidimensional M es
% suficientemente similar al color que indican las componentes del vector c,
% resultado de la operación 'impixel'.

function a = proximo(M,c,margen,i,j,k)

if (abs(double(M(i,j,k))-double(c(k)))<=margen)
  a=1;
else
  a=0;
end
return
```

4.3 ALGORITMO DE VALIDACIÓN DEFINITIVO

4.3.1 INTRODUCCIÓN

A continuación se describirá el algoritmo definitivo utilizado para el proceso de validación de los programas de segmentación. Dicho algoritmo es el resultante de notables mejoras realizadas en el primero de ellos (punto 3.3).

El método seguido se denomina EM (Espectation-Maximization). Calcula una imagen que representa la segmenteción de máxima verosimilitud (donde cada píxel es una variable aleatoria binaria) y también realiza una caracterización numérica de las prestaciones de cada decisión (expertos y programa de segmentación).

Para profundizar en el desarrollo matemático que da lugar a las expresiones que seguidamente se tratan, es recomendable la lectura del correspondiente artículo (ver anexo nº2), creado por los mismos autores que el anterior (que fue desestimado como solución válida).

4.3.2 FUNCIONAMIENTO DEL ALGORITMO

El funcionamiento del algoritmo es parecido al del anterior, es decir, consiste en un proceso iterativo de cálculos de probabilidad (los llamados *parámetros de calidad*) y de imágenes a las que se ha dado el nombre de *matrices de pesos* o *matrices de variables de peso*.

A pesar de seguir estrategias similares, las expresiones utilizadas son diferentes a las empleadas en el algoritmo anterior. Son el resultado de un estudio bastante más elaborado.

Por otra parte, la matriz de pesos no será una imagen binaria, sino que representará, haciendo uso de una escala de grises, la probabilidad de que un determinado píxel sea considerado como perteneciente a la región significativa, o, lo que es equivalente, la probabilidad de que la imagen de máxima verosimilitud valga 1 en dicho píxel.

Las condiciones de partida del algoritmo no serán las mismas. En el anterior, era preciso realizar una estimación de la imagen de máxima verosimilitud inicial, decidiendo 1 ó 0 por mayoría; sin embargo, en este caso se considerará como solución aceptable el partir de unos determinados parámetros de calidad (normalmente valdrán todos, al inicio, 0.9). Con dichos parámetros se calculará una matriz de pesos, que originará a su vez un nuevo valor de los mismos. Esto se repite hasta alcanzar la convergencia.

4.3.2.1 Cálculo de la matriz de pesos

En el desarrollo seguido (ver anexo nº 2), W_i hace referencia al valor del i-ésimo píxel de la matriz de variables de peso. La expresión con la que se calcula dicho valor a partir de los parámetros de calidad es:

$$W_i = \frac{g(T_i = 1)a}{g(T_i = 1)a + (1 - g(T_i = 1))b}$$

donde:

$$a = \prod_{j:D_{ij}=1} p_j \prod_{j:D_{ij}=0} (1 - p_j) \qquad b = \prod_{j:D_{ij}=0} q_j \prod_{j:D_{ij}=1} (1 - q_j)$$

NOTA: $j: D_{ij} = X$ indica: Para todas las imágenes de entrada (experto o algoritmo j-ésimo) en las que se ha decidido que el valor del píxel i-ésimo sea X.

$g(T_i = 1)$ es la probabilidad a priori de que el píxel i-ésimo de la imagen de máxima verosimilitud sea 1. Con este valor de entrada, que hace de ponderación, se establece la susceptibilidad del algoritmo a marcar píxeles como pertenecientes a la región significativa de la imagen en cuestión, es decir, cuanto más alta sea dicha probabilidad, más claros serán los puntos de la matriz de pesos. De lo contrario, si se establece un valor pequeño, la matriz de pesos dará lugar a una imagen bastante más selectiva, contribuyendo a la correspondiente alteración de los parámetros de calidad.

El valor que se debe imponer para g(T=1) debe ser similar a la fracción del área total de la imagen que ocupa la región significativa. Como existen tantas regiones significativas como número de sujetos, se efectuará la media de entre todas las proporciones obtenidas (esta operación requerirá un notable tiempo de ejecución extra, dado el gran tamaño de las imágenes).

El programa de MATLAB diseñado para calcular la matriz W_i es:

-- MATRIZ_PESOS.M --

```
% Programa que calcula la matriz de pesos W a partir de las imágenes
% binarias de entrada, la ponderación g(T) y los parámetros de calidad


function W = matriz_pesos(A,p,q,g)

alfa=1;
beta=1;

for i=1:size(A,1)
    for j=1:size(A,2)

        for k=1:size(A,3)

             % A(i,j,k) corresponde al píxel (i,j) de la imagen del
             % experto o algoritmo j-ésimo.

            alfa = alfa*(p(1,k)*(A(i,j,k)==1)+(1-p(1,k))*(A(i,j,k)==0) );
            beta = beta*(q(1,k)*(A(i,j,k)==0)+(1-q(1,k))*(A(i,j,k)==1) );

        end

        % Expresión 4.3.2.1

        W(i,j) = (g * alfa) / ( (g*alfa) + ((1-g)*beta) );

        alfa = 1;
        beta = 1;

    end
end

return
```

A continuación se facilita también el código de la función encargada de calcular el valor de g(T=1). Este cálculo automático de g(T=1) sólo se efectúa en la última versión del programa (*calculaG.m* no existe en la primera de las versiones, por lo que el usuario debe de introducir el valor que, a simple vista, crea conveniente):

-- CALCULAG.M –

```
% Funcion que calcula el factor de ponderacion g(T=1)

function g = calculaG(A)

unos = 0;
ceros = 0;
suma = 0;
```

```
for k=1:size(A,3)

    for i=1:size(A,1)
        for j=1:size(A,2)


            % Calculo del numero de pixeles de valor 1 en la imagen k:
            if A(i,j,k)==1
                unos = unos +1;
            end

            % Calculo del numero de pixeles de valor 0 en la imagen k:
            if A(i,j,k)==0
                ceros = ceros +1;
            end

            % Relacion unos-ceros en imagen k:
            proporcion(1,k) = unos/(unos + ceros);

        end
    end

    unos = 0;
    ceros = 0;

end

% Para g(T=1) se escoge la media de  las relaciones unos-ceros de todas las
% imagenes:

for k=1:size(A,3)
    suma = suma + proporcion(1,k);
end

% Valor de g(T=1):
g = suma/size(A,3);

return
```

4.3.2.2 Cálculo de los parámetros de calidad

Al igual que en el segundo algoritmo de validación (apartado 3.3), se calculan unas probabilidades llamadas P y Q. No obstante, según el anexo nº2, el significado de las probabilidades es contrario al del algoritmo actual, por lo que seguramente existía una errata en el anterior. Es decir: P (también llamada *sensibilidad*) es la probabilidad de que en una imagen se marquen píxeles con valor 1 cuando, en efecto, también son marcados con dicho valor en la imagen de máxima verosimilitud, y Q (también llamada *especificidad*) se definirá de igual forma para el valor 0.

La base del desarrollo del que se obtienen las expresiones para el cálculo de los parámetros de calidad consiste en maximizar el argumento de la función densidad de probabilidad *f(D,T|p,q)*, donde p y q son los parámetros de calidad, D es la imagen correspondiente a la decisión de un determinado sujeto y T es la imagen de máxima verosimilitud. El planteamiento principal del problema es:

$$\hat{p},\hat{q} = \arg\max_{p,q}\left[\ln\ f(D,T \mid p,q)\right]$$

Tras el correspondiente desarrollo, las expresiones que permiten obtener el valor de las probabilidades a partir de la matriz de pesos (W_i) y de las decisiones que constan en las imágenes de entrada (D_{ij}) son:

$$p_j = \frac{\sum_{i:D_{ij}=1} W_i}{\sum_{i:D_{ij}=1} W_i + \sum_{i:D_{ij}=0} W_i} \qquad q_j = \frac{\sum_{i:D_{ij}=0} (1-W_i)}{\sum_{i:D_{ij}=1} (1-W_i) + \sum_{i:D_{ij}=0} (1-W_i)}$$

Programa en MATLAB para el cálculo de los parámetros:

-- CALCULAP_PESOS.M --

```
% Programa que obtiene el valor de los parámetros de calidad para cada experto
% y/o algoritmo a partir de las imágenes binarias de entrada y de la matriz de
% pesos W


function [p,q] = calculaP_pesos(A,W)

% Sumatorios

sum_p1=0;
sum_p2=0;

sum_q1=0;
sum_q2=0;

% Para cada experto/algoritmo k:
for k=1:size(A,3)

    for i=1:size(A,1)
        for j=1:size(A,2)

            sum_p1 = sum_p1 + W(i,j)*(A(i,j,k)==1);
            sum_p2 = sum_p2 + W(i,j)*(A(i,j,k)==0);

            sum_q1 = sum_q1 + (1-W(i,j))*(A(i,j,k)==0);
            sum_q2 = sum_q2 + (1-W(i,j))*(A(i,j,k)==1);
        end
    end
```

```
% Expresión 4.3.2.2

    p(1,k) = sum_p1 / ( sum_p1 + sum_p2 );

    q(1,k) = sum_q1 / ( sum_q1 + sum_q2 );

    sum_p1=0;
    sum_p2=0;

    sum_q1=0;
    sum_q2=0;
end

return
```

4.3.2.3 Cálculo del DSC (Dise Similarity Coeficient)

El DSC proporciona resultados aceptables en cuanto a la similitud entre contornos; no obstante, presenta inconvenientes que lo hacen un mecanismo menos robusto y fiable que la solución tomada.

En el software diseñado, además de la representación de los parámetros de calidad, se proporcionarán los valores del DSC entre cada experto y la matriz de pesos final. Existen casos en los que el DSC de dos expertos es muy parecido, a diferencia de lo que ocurre con los parámetros de calidad. Por tanto, se puede afirmar que la solución tomada proporciona una mayor información.

El cálculo de este coeficiente se realiza de la siguiente forma:

$$DSC = \frac{2\left|A \cap B\right|}{\left|A\right| + \left|B\right|}$$

donde |A| es el área o número de píxeles de la región A (región resultante de la decisión de un experto o de la segmentación del algoritmo automático a verificar) y |B| el área que comprenden los píxeles de la matriz de pesos cuyo valor supera 0.5.

Lógicamente, $|A \cap B|$ corresponde al área o número de píxeles de la intersección entre las regiones A y B.

A continuación se detalla el programa en MATLAB que calcula el DSC:

-- CALCULADSC.M --

```
% Programa que calcula el Dice Similarity Coeficient a partir de las
% imágenes binarias de entrada y la matriz de pesos.


function DSC = calculaDSC(A,W)


% Para cada experto o algoritmo

for k=1:size(A,3)

    inter = 0;
    cont_A = 0;
    cont_W = 0;

    % Se recorre cada imagen binaria

    for i=1:size(W,1)
        for j=1:size(W,2)

            % Cálculo de la intersección:

            if (A(i,j,k)==1 & W(i,j)>=0.5)
                inter = inter + 1;
            end


            % Área de A:

            if A(i,j,k)==1
                cont_A = cont_A + 1;
            end


            % Área de B:

            if W(i,j)>=0.5
                cont_W = cont_W + 1;
            end

        end
    end


    % Expresión 4.3.2.3:

    DSC(1,k) = 2*inter/(cont_A + cont_W);

end
```

4.3.2.4 Condición de parada del algoritmo

Dado que las soluciones anteriores fueron descartadas, se ha decidido tratar los diversos aspectos complementarios en el diseño del programa únicamente en el apartado que corresponde a la solución definitiva. Entre estos aspectos se halla el problema de la finalización del algoritmo.

Como se ha comentado, el algoritmo de validación es de carácter iterativo. Hasta ahora, lo único que se ha explicado acerca de esto último es que el fin de la ejecución se producirá una vez que los parámetros de calidad permanezcan inalterables de una iteración a otra. Sin embargo, existen casos en los que conviene establecer condiciones de parada adicionales, principalmente cuando se tiene un gran número de imágenes de entrada o grandes contornos a analizar, ya que los cálculos requieren mucho tiempo de ejecución.

Las condiciones adicionales son:

- Fijar el *número exacto de iteraciones* que se van a ejecutar. Como es lógico, si se llega a la convergencia exacta antes de que transcurran las iteraciones indicadas también se detendrá la ejecución y se mostrarán los resultados, para así ahorrar tiempo.

- Fijar una diferencia máxima entre los valores de un parámetro de calidad correspondientes a dos iteraciones consecutivas (no compatible con la anterior condición). Esta diferencia se puede denominar *margen de convergencia*.

- Independientemente de las anteriores, fijar un *tiempo máximo de ejecución* del programa, por si las anteriores condiciones resultan ser difícilmente alcanzables.

Estas condiciones serán incluidas en la última versión del programa.

4.4 PROGRAMA PRINCIPAL: PRIMERA VERSIÓN

Finalmente queda por detallar el programa principal que comprende todos los programas anteriores. Dicho programa es el siguiente:

-- SVAS1.M --

```
% Primera versión del Software para la Validación de Algoritmos de Segmentación


function [p,q] = svas1(varargin)

repetir=1;


%      CAPTURA DE LAS IMÁGENES EN ARRAY MULTIDIMENSIONAL B
%-----------------------------------------------------------------

k=1;

for k=1:nargin-1
   B(:,:,:,k) = imread(varargin{k});
end


%     MODIFICACIONES en la imagen del algoritmo automático.
%     Es obligatorio ponerlo como último argumento de entrada.
%-----------------------------------------------------------------

aux = imread(varargin{nargin});
B(:,:,:,nargin) = prepara_alg(aux,size(B(:,:,:,1)));

%-----------------------------------------------------------------



%     SELECCIÓN DE REGIÓN Y DETECCIÓN DEL COLOR
%-----------------------------------------------------------------


% Se almacena en RECT las coordenadas del rectángulo de selección
% realizado en la primera imagen manualmente.

[X,Y,C(:,:,:,1),RECT] = imcrop(B(:,:,:,1));

% La selección en el resto de imágenes será idéntica a la anterior

for k=2:nargin
   C(:,:,:,k) = imcrop(B(:,:,:,k),RECT);
end
```

```
% Especificacion del color a detectar en cada imagen:

for k=1:nargin

     % Para facilitar pinchar en el contorno:
    aux = imcrop(C(:,:,:,k));

    % Captura de las componentes RGB del pixel seleccionado:
    color(k,:) = impixel(aux);
end


% Detección del color:

for k=1:nargin
    A(:,:,k) = detecta_color(C(:,:,:,k),color(k,:),20);
end


% Opción de repetir con nuevo valor de g(T=1), sin tener que volver a seleccionar
% regiones y colores en las imágenes originales:


while repetir

   % Introducir valor de g(T=1) (ponderación)
   G = input('Introduce g(Ti=1): ');

   % Se rellena el interior de los contornos:

   for k=1:nargin
        A(:,:,k) = bwfill(A(:,:,k),'holes');
        close;
   end


    % Inicialización de los parámetros:

   for k=1:size(A,3)
      p(1,k) = 0.9;
      q(1,k) = 0.9;
   end


   % Se sacan en la ventana de comandos de MATLAB los parámetros de
   % calidad:

   imprimirPQ(A,p,q);

    % Cálculo de la matriz de pesos

   [W] = matriz_pesos(A,p,q,G);
   figure,imshow(W),title('T inicial');

   flag = 1;
   cont = 0;
```

```
%-----------------------------------------------------------------
%     PARTE ITERATIVA DEL ALGORITMO
%-----------------------------------------------------------------

   while flag


    % Cálculo de los parámetros de calidad y de la matriz de pesos

      p_ant = p;
      q_ant = q;

      [p,q] = calculaP_pesos(A,W);
      [W] = matriz_pesos(A,p,q,G);

      disp('Nueva matriz T:');
      figure,imshow(W),title('T nueva');

      imprimirPQ(A,p,q);

      % Se comprueba si se ha alcanzado la convergencia:

      if p_ant == p & q_ant == q
          flag=0;
          disp('Finalizando el algoritmo...');
      end

   end

     % Se pregunta si se desea calcular el DSC:
   if input('Calcular DSC -> 1   No calcular -> 0: ')
        DSC = calculaDSC(A,W);
   end


   % Se pregunta si se desea ejecutar de nuevo el programa:
   repetir = input('Repetir con otras opciones -> 1  Finalizar -> 0: ');

end

return
```

-- IMPRIMIRPQ.M –

```
% Programa que representa en la línea de comandos de MATLAB los valores
% de los parámetros de calidad.

function imprimirPQ(A,p,q)

       disp('Probabilidades:');
       disp(sprintf(' Tabla de P:\n      '));

       for i=1:size(A,3)
           disp(sprintf(' %f ',p(1,i)));
       end
       disp(sprintf('\n\n Tabla de Q:\n     '));
       for i=1:size(A,3)
           disp(sprintf(' %f ',q(1,i)));
       end
```

Instrucciones de ejecución:

Para ejecutar el programa se ha de escribir su nombre en la línea de comandos. Como argumentos han de ser introducidos, entre comas, los nombres y extensión de los archivos que contienen las correspondientes imágenes:

```
>> [p,q] = svas1('expert1.bmp','expert2.bmp', 'expert3.bmp',..., 'alg.jpg');
```

Es importante tener en cuenta que el último argumento debe ser el nombre de la imagen proporcionada por el programa automático de segmentación, ya que sobre éste se ejecutará el programa *prepara_alg* . Si no se necesita este programa (ver 4.2.3) el orden es indiferente, y se pueden alternar decisiones de expertos o de programas de segmentación sin ningún problema.

Durante la ejecución se irán visualizando tanto los resultados numéricos como los gráficos, y se da la opción al usuario de repetir el análisis con una configuración diferente, sin tener que volver a especificar una región de trabajo o los colores a identificar.

La única ventaja que presenta esta primera versión es que el número de imágenes de entrada no está limitado, cosa que sí ocurre en la versión posterior, aunque, como a continuación se verá, ésta última está dotada de muchas más prestaciones.

4.5 SOFTWARE PARA LA VALIDACIÓN DE ALGORITMOS DE SEGMENTACIÓN

4.5.1 INTRODUCCIÓN

Es evidente que todo programa que se ejecuta por medio de un entorno gráfico ofrece un gran número de prestaciones y permite una muy cómoda maniobrabilidad. Teniendo esto en cuenta, se ha decidido crear dicha interfaz para llevar a cabo la ejecución de la versión definitiva del programa, obteniéndose un software más manejable que el correspondiente a la versión anterior, que requería la ejecución desde la línea de comandos de MATLAB.

Además de transportar los programas anteriores a un entorno visual, han sido creadas nuevas funciones y realizadas diversas mejoras que se explicarán en lo sucesivo.

Antes de nada, se detalla un diagrama de la ventana de trabajo, es decir, del interfaz visual a través del cual se solicitarán las diversas operaciones:

4.5.2 DESCRIPCIÓN GENERAL DE LA INTERFAZ

4.5.2.1 Componentes

Todas las variables ubicadas bajo fondo azul hacen referencia a los parámetros que debe introducir el usuario, tales como los nombres de archivos de imagen, condiciones iniciales, de parada, y otras opciones. En gris se muestran los casilleros correspondientes a los resultados que facilitará el programa tras su ejecución.

Introducción de archivos y visualización de resultados numéricos:

En la parte inferior se pueden observar una serie de casilleros; en los de la fila superior se introducen los nombres (ruta) de los archivos que incluyen cada una de las imágenes de entrada a procesar (se tendrán en cuenta para el procesado si la opción correspondiente está activada).

Tras varias operaciones, y teniendo en cuenta el resto de opciones de configuración, se obtienen los valores de los parámetros P, Q y DSC para cada experto o programa de segmentación. Estos resultados se mostrarán en los casilleros de las filas restantes.

Un pequeño inconveniente que surge al disponer del entorno gráfico es que hay que fijar el número de casilleros correspondientes a cada sujeto. Se ha considerado suficiente contar con ocho (cinco expertos y 3 segmentaciones, aunque es posible alterar esta distribución a gusto del usuario, siempre y cuando las imágenes relativas a los procesos de segmentación no necesiten preprocesado -ver 4.2.3-, que es para lo que se han reservado estos tres últimos casilleros).

Captura y visualización de las imágenes:

Para comprobar que no existen errores, es conveniente ofrecer la posibilidad de visualizar, en cualquier momento, las imágenes seleccionadas anteriormente. Para esto, se debe pulsar, una vez escritos los nombres de los ficheros de entrada, el botón CAPTURAR, ubicado en la parte superior derecha de la ventana, y que creará una lista con las imágenes seleccionadas.

Configuración:

Bajo el botón anterior se hallan diversos casilleros correspondientes a aspectos de notable importancia a la hora de ejecutar el algoritmo de validación.

Se deben detallar, por un lado, los valores de los parámetros de calidad iniciales (***Pini*** y ***Qini***). Por defecto, el valor inicial de dichos parámetros será 0.9.

También es necesario establecer la condición de parada, bien sea por número máximo de iteraciones (***Nº iteraciones***) o por error máximo (***Error*** será la escogida por defecto, con un valor de 10^{-6}). Independientemente, se debe introducir, por si las condiciones de parada resultan ser difícilmente alcanzables, un *límite de tiempo de ejecución* una vez comenzadas las iteraciones del algoritmo de validación (suele ser de unos 5 minutos, o mayor para imágenes de gran tamaño).

Histórico de resultados:

Dado que en la tabla de casilleros inferior se muestran únicamente los valores finales de los parámetros de calidad y del coeficiente de Dice (DSC), se ha considerado conveniente crear un fichero de texto (*iteraciones.txt*) que incluya, ordenadamente, los valores de dichos parámetros una vez transcurrida cada una de las iteraciones, por si el usuario desea observar la evolución de los mismos hasta alcanzar la convergencia o cumplir alguna de las condiciones de parada.

No obstante, si no se desea buscar el archivo y abrirlo, el software ofrece también la posibilidad de que ese mismo contenido sea escrito en la ventana de comandos de MATLAB, con tan sólo activar la opción ***Extraer todos los valores***.

Otros resultados de interés:

Tras la finalización, se mostrarán en los casilleros correspondientes el número de iteraciones que han sido necesarias (***Iteraciones***), la mayor diferencia existente entre un parámetro de salida y el de la iteración anterior (***Error***), así como el valor ***g(T=1)*** óptimo para el ejemplo que se considere.

Además de los resultados numéricos, es conveniente almacenar los resultados gráficos, tales como las matrices de pesos inicial y final (***Matrices Pesos****: mat_original.bmp, mat_final.bmp*), y también las imágenes binarias resultantes de la selección del color para cada experto o algoritmo (***Contornos****: seleccion1.bmp, seleccion2.bmp, ...*). Éstas últimas pueden resultar útiles para comprobar los resultados o para iniciar el algoritmo con una configuración diferente para así evitar tener que repetir la selección de subregión y color en cada una de las imágenes, tareas que suponen un considerable tiempo de ejecución.

4.5.2.2 Instrucciones principales de funcionamiento

Para abrir el programa se ha de teclear ***svas*** en la ventana de MATLAB. Su utilización por parte del usuario debería resultar bastante sencilla una vez leídos los puntos anteriores.

Al iniciar, y una vez finalizado el programa, conviene pulsar RESET, para reestablecer los valores por defecto y eliminar los resultados anteriores.

Una vez configurada la interfaz e introducidos los nombres de los archivos de imágenes, conviene pulsar CAPTURAR para visualizarlas y comprobar que no existen errores (si son las imágenes deseadas, todas tienen el mismo tamaño, etc.). Seguidamente se ha de pulsar EJECUTAR para que el programa comience a procesarlas y a realizar la validación, exigiendo las acciones correspondientes al usuario (selección de subregiones, detección del color, etc) a través de una línea de texto ubicada bajo la imagen.

Cuando el programa muestra al fin los resultados numéricos, el usuario puede pulsar RESET y comenzar otra prueba diferente, o pulsar REPETIR para volver a ejecutarlo con los mismos contornos, pero con una nueva configuración (y así ahorrar el tiempo de volver a seleccionar regiones y colores, que suele ser de varios minutos para las imágenes tratadas en los ejemplos).

Si se desea que el programa se inicie desde imágenes ya preprocesadas en ejecuciones anteriores, basta con escribir el nombre de cada una de ellas y activar la opción ***partir de imágenes binarias***, evitándose también así todo el proceso de tratamiento de las mismas.

4.5.3 CÓDIGO DEL PROGRAMA PRINCIPAL

A continuación se detalla el código del programa principal, comentado debidamente. El hecho de diseñar una interfaz para el algoritmo de validación que ofrezca un gran número de prestaciones conlleva la creación de programas cuyo código es bastante extenso, tal y como ocurre con el programa ***interface.m.***

- INTERFACE.M -

```
%==========================================================================
%  SOFTWARE PARA LA VALIDACION DE ALGORITMOS DE SEGMENTACION.
%     (ENTORNO VISUAL)
%==========================================================================

function varargout = interface(varargin)

% interface M-file for interface.fig

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @interface_OpeningFcn, ...
                   'gui_OutputFcn',  @interface_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 interface_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;
guidata(hObject, handles);

imshow('presentacion.bmp');
warning off MATLAB:MKDIR:DirectoryExists;
mkdir resultados;
set(handles.errorflag,'Value',1);

function varargout = svas_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

```
%--------------------------------------------------------------------------------
%   CONTROL DEL VALOR DE ENTRADA g(T=1)
%--------------------------------------------------------------------------------

function G_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function G_Callback(hObject, eventdata, handles)

% Captura del valor especificado:

G = str2double(get(hObject, 'String'));

if isnan(G)
    set(hObject, 'String', 0.5);
    errordlg('Input must be a number','Error');
end

% Almacenamiento del valor:

data = getappdata(gcbf, 'metricdata');
data.G = G;
setappdata(gcbf, 'metricdata', data);



%--------------------------------------------------------------------------
%   CONDICIONES PARA FIN DEL ALGORITMO
%--------------------------------------------------------------------------

% Opcion de especificacion de error maximo permisible:

function errorflag_Callback(hObject, eventdata, handles)

if get(hObject,'value')
    aux = 1;

    % Desactivacion de la opcion alternativa:

    set(handles.Niterflag, 'Value', 0);
    set(handles.Niter,'BackgroundColor',[0.5020 0.5020 1.0000],'enable','off');

    % Activacion del casillero para especificacion del error:

    set(handles.error, 'BackgroundColor','white','enable','on');
else
    aux = 0;

    % Deshabilitacion del casillero si se desactiva la correspondiente opcion:

    set(handles.error,'BackgroundColor',[0.5020 0.5020 1.0000],'enable','off');
end
```

```
% Se establece el valor binario de errorflag para su uso futuro:

data = getappdata(gcbf, 'metricdata');
data.errorflag = aux;
setappdata(gcbf, 'metricdata', data);

% Linea de indicaciones de la interfaz:

set(handles.texto,'String','Escribir un valor para el maximo error permitido');


% Opcion de especificacion del numero maximo de iteraciones permisible:

function Niterflag_Callback(hObject, eventdata, handles)

if get(hObject,'value')
    aux = 1;

    % Desactivacion de la opcion alternativa:

    set(handles.errorflag, 'Value', 0);
    set(handles.error,'BackgroundColor',[0.5020 0.5020 1.0000],'enable',
                                                                    'off');

    % Activacion del casillero para especificacion del numero de iteraciones:

    set(handles.Niter, 'BackgroundColor','white','enable','on');
else
    aux = 0;

    % Deshabilitacion del casillero si se desactiva la correspondiente opcion:

    set(handles.Niter,'BackgroundColor',[0.5020 0.5020 1.0000],'enable','off');
end

% Se establece el valor binario de Niterflag para su uso futuro:

data = getappdata(gcbf, 'metricdata');
data.Niterflag = aux;
setappdata(gcbf, 'metricdata', data);

% Linea de indicaciones de la interfaz:

set(handles.texto,'String','Escribir el numero maximo de iteraciones a
realizar');



%--------------------------------------------------------------------------
%   ESPECIFICACION DEL ERROR MAXIMO Y NUMERO DE ITEREACIONES
%--------------------------------------------------------------------------

% Error maximo:

function error_CreateFcn(hObject, eventdata, handles)
if ispc

    % En este caso y en los sucesivos se elimina la linea de establecimiento
    % del color (blanco) para los casilleros, para poder asi modificarlos
    % libremente:

    %set(hObject,'BackgroundColor','white');
```

```
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function error_Callback(hObject, eventdata, handles)

% Captura del valor del error especificado:

error = str2double(get(hObject, 'String'));
if isnan(error)
    set(hObject, 'String', 0.01);
    errordlg('Input must be a number','Error');
end

% Almacenamiento del valor para su uso futuro:

data = getappdata(gcbf, 'metricdata');
data.error = error;
setappdata(gcbf, 'metricdata', data);



% Numero de iteraciones:

function Niter_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Niter_Callback(hObject, eventdata, handles)
    Niter = str2double(get(hObject, 'String'));
    if isnan(Niter)
        set(hObject, 'String', 100);
        errordlg('Input must be a number','Error');
    end

% Almacenamiento del valor para su uso futuro:

data = getappdata(gcbf, 'metricdata');
data.Niter = Niter;
setappdata(gcbf, 'metricdata', data);



%--------------------------------------------------------------------------------
%   ESPECIFICACION DE LOS ARCHIVOS QUE CONTIENEN LAS IMAGENES DE ENTRADA
%--------------------------------------------------------------------------------

% Dada la analogia, unicamente se escriben comentarios en el primer ejemplo.
% ------------------------------------------------------------------------


function archivo1_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function archivo1_Callback(hObject, eventdata, handles)

% Captura del nombre del archivo (RUTA):

archivo1 = str2double(get(hObject, 'String'));

% Almacenamiento en memoria de dicho nombre:

data = getappdata(gcbf, 'metricdata');
data.archivo1 = archivo1;
setappdata(gcbf, 'metricdata', data);


function archivo2_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function archivo2_Callback(hObject, eventdata, handles)
archivo2 = str2double(get(hObject, 'String'));

data = getappdata(gcbf, 'metricdata');
data.archivo2 = archivo2;
setappdata(gcbf, 'metricdata', data);


function archivo3_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function archivo3_Callback(hObject, eventdata, handles)
archivo3 = str2double(get(hObject, 'String'));

data = getappdata(gcbf, 'metricdata');
data.archivo3 = archivo3;
setappdata(gcbf, 'metricdata', data);


function archivo4_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function archivo4_Callback(hObject, eventdata, handles)
archivo4 = str2double(get(hObject, 'String'));

data = getappdata(gcbf, 'metricdata');
data.archivo4 = archivo4;
setappdata(gcbf, 'metricdata', data);
```

```
function archivo5_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function archivo5_Callback(hObject, eventdata, handles)
archivo5 = str2double(get(hObject, 'String'));

data = getappdata(gcbf, 'metricdata');
data.archivo5 = archivo5;
setappdata(gcbf, 'metricdata', data);



function archivoA_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function archivoA_Callback(hObject, eventdata, handles)
archivoA = str2double(get(hObject, 'String'));

data = getappdata(gcbf, 'metricdata');
data.archivoA = archivoA;
setappdata(gcbf, 'metricdata', data);



function archivoA2_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function archivoA2_Callback(hObject, eventdata, handles)
archivoA2 = str2double(get(hObject, 'String'));

data = getappdata(gcbf, 'metricdata');
data.archivoA2 = archivoA2;
setappdata(gcbf, 'metricdata', data);



function archivoA3_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function archivoA3_Callback(hObject, eventdata, handles)
archivoA3 = str2double(get(hObject, 'String'));

data = getappdata(gcbf, 'metricdata');
data.archivoA3 = archivoA3;
setappdata(gcbf, 'metricdata', data);
```

```
%--------------------------------------------------------------------------------
%   CASILLEROS DE PROBABILIDADES Y DSC
%--------------------------------------------------------------------------------

% CONFIGURACION DE PARAMETROS DE CALIDAD INICIALES (Pini Y Qini):

function Pini_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Pini_Callback(hObject, eventdata, handles)
    Pini = str2double(get(hObject, 'String'));
    if isnan(Pini)
        set(hObject, 'String', 0.9);
        errordlg('Input must be a number','Error');
    end

% Almacenamiento del valor:

data = getappdata(gcbf, 'metricdata');
data.Pini = Pini;
setappdata(gcbf, 'metricdata', data);



function Qini_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Qini_Callback(hObject, eventdata, handles)
    Qini = str2double(get(hObject, 'String'));
    if isnan(Qini)
        set(hObject, 'String', 0.9);
        errordlg('Input must be a number','Error');
    end

% Almacenamiento del valor:

data = getappdata(gcbf, 'metricdata');
data.Qini = Qini;
setappdata(gcbf, 'metricdata', data);



% ----------------------------------------------------------------
% CASILLERO DE SALIDA PARA LOS PARAMETROS RESULTANTES: P, Q Y DSC
% ----------------------------------------------------------------


function p1_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function p1_Callback(hObject, eventdata, handles)
```

```
function p2_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function p2_Callback(hObject, eventdata, handles)



function p3_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function p3_Callback(hObject, eventdata, handles)



function p4_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function p4_Callback(hObject, eventdata, handles)



function p5_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function p5_Callback(hObject, eventdata, handles)



function Palg_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Palg_Callback(hObject, eventdata, handles)



function Palg2_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
function Palg2_Callback(hObject, eventdata, handles)
```

```
function Palg3_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Palg3_Callback(hObject, eventdata, handles)


%==============================


function q1_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function q1_Callback(hObject, eventdata, handles)



function q2_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function q2_Callback(hObject, eventdata, handles)



function q3_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function q3_Callback(hObject, eventdata, handles)



function q4_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function q4_Callback(hObject, eventdata, handles)


function q5_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function q5_Callback(hObject, eventdata, handles)


function Qalg_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Qalg_Callback(hObject, eventdata, handles)


function Qalg2_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Qalg2_Callback(hObject, eventdata, handles)


function Qalg3_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function Qalg3_Callback(hObject, eventdata, handles)


%==============================


function dsc1_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function dsc1_Callback(hObject, eventdata, handles)


function dsc2_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function dsc2_Callback(hObject, eventdata, handles)
```

```
function dsc3_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function dsc3_Callback(hObject, eventdata, handles)



function dsc4_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function dsc4_Callback(hObject, eventdata, handles)



function dsc5_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function dsc5_Callback(hObject, eventdata, handles)



function DSCalg_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function DSCalg_Callback(hObject, eventdata, handles)



function DSCalg2_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function DSCalg2_Callback(hObject, eventdata, handles)


function DSCalg3_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function DSCalg3_Callback(hObject, eventdata, handles)
```

```
%--------------------------------------------------------------------------------
%   RESULTADOS DEL ERROR Y NUMERO DE ITERACIONES OBTENIDOS
%--------------------------------------------------------------------------------


function NiterFINAL_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function NiterFINAL_Callback(hObject, eventdata, handles)



function errorFINAL_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function errorFINAL_Callback(hObject, eventdata, handles)



%--------------------------------------------------------------------------------
%   OPCIONES DE SELECCION DE EXPERTOS/ALGORITMOS
%--------------------------------------------------------------------------------

% Unicamente se comenta el primero (por analogia)
% ---------------------------------------------

function boolean1_Callback(hObject, eventdata, handles)

if get(hObject,'value')
    aux = 1;

    % Habilitacion del casillero para escribir ruta y nombre del archivo:

    set(handles.archivo1,'enable','on');

    % Modificacion de los colores de los casilleros para dar sensacion de
    % activacion:

    set(handles.archivo1,'BackgroundColor','white');
    set(handles.p1,'BackgroundColor','white');
    set(handles.q1,'BackgroundColor','white');
    set(handles.dsc1,'BackgroundColor','white');
else
    aux = 0;

    % Deshabilitacion de los casilleros si se desactiva la opcion:

    set(handles.archivo1,'enable','off');
    set(handles.archivo1,'BackgroundColor',[0.5020 0.5020 1.0000]);
    set(handles.p1,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.q1,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.dsc1,'BackgroundColor',[0.5020 0.5020 0.5020]);

end
```

```
% Almacenamiento del valor binario para su uso futuro:

data = getappdata(gcbf, 'metricdata');
data.boolean1 = aux;
setappdata(gcbf, 'metricdata', data);

% Activacion/Desactivacion del boton EJECUTAR segun existan o no ficheros
% de entrada:

if alguno_ON(gcbf,handles)==0
    set(handles.run,'enable','off');
else
    set(handles.run,'enable','on');
end


function boolean2_Callback(hObject, eventdata, handles)
if get(hObject,'value')
    aux = 1;
    set(handles.archivo2,'enable','on');
    set(handles.archivo2,'BackgroundColor','white');
    set(handles.p2,'BackgroundColor','white');
    set(handles.q2,'BackgroundColor','white');
    set(handles.dsc2,'BackgroundColor','white');

else
    aux = 0;
    set(handles.archivo2,'enable','off');
    set(handles.archivo2,'BackgroundColor',[0.5020 0.5020 1.0000]);
    set(handles.p2,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.q2,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.dsc2,'BackgroundColor',[0.5020 0.5020 0.5020]);
end

data = getappdata(gcbf, 'metricdata');
data.boolean2 = aux;
setappdata(gcbf, 'metricdata', data);

if alguno_ON(gcbf,handles)==0
    set(handles.run,'enable','off');
else
    set(handles.run,'enable','on');
end


function boolean3_Callback(hObject, eventdata, handles)
if get(hObject,'value')
    aux = 1;
    set(handles.archivo3,'enable','on');
    set(handles.archivo3,'BackgroundColor','white');
    set(handles.p3,'BackgroundColor','white');
    set(handles.q3,'BackgroundColor','white');
    set(handles.dsc3,'BackgroundColor','white');
else
    aux = 0;
    set(handles.archivo3,'enable','off');
    set(handles.archivo3,'BackgroundColor',[0.5020 0.5020 1.0000]);
    set(handles.p3,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.q3,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.dsc3,'BackgroundColor',[0.5020 0.5020 0.5020]);

end
```

```
data = getappdata(gcbf, 'metricdata');
data.boolean3 = aux;
setappdata(gcbf, 'metricdata', data);

if alguno_ON(gcbf,handles)==0
    set(handles.run,'enable','off');
else
    set(handles.run,'enable','on');
end


function boolean4_Callback(hObject, eventdata, handles)
if get(hObject,'value')
    aux = 1;
    set(handles.archivo4,'enable','on');
    set(handles.archivo4,'BackgroundColor','white');
    set(handles.p4,'BackgroundColor','white');
    set(handles.q4,'BackgroundColor','white');
    set(handles.dsc4,'BackgroundColor','white');
else
    aux = 0;
    set(handles.archivo4,'enable','off');
    set(handles.archivo4,'BackgroundColor',[0.5020 0.5020 1.0000]);
    set(handles.p4,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.q4,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.dsc4,'BackgroundColor',[0.5020 0.5020 0.5020]);
end

data = getappdata(gcbf, 'metricdata');
data.boolean4 = aux;
setappdata(gcbf, 'metricdata', data);

if alguno_ON(gcbf,handles)==0
    set(handles.run,'enable','off');
else
    set(handles.run,'enable','on');
end



function boolean5_Callback(hObject, eventdata, handles)
if get(hObject,'value')
    aux = 1;
    set(handles.archivo5,'enable','on');
    set(handles.archivo5,'BackgroundColor','white');
    set(handles.p5,'BackgroundColor','white');
    set(handles.q5,'BackgroundColor','white');
    set(handles.dsc5,'BackgroundColor','white');
else
    aux = 0;
    set(handles.archivo5,'enable','off');
    set(handles.archivo5,'BackgroundColor',[0.5020 0.5020 1.0000]);
    set(handles.p5,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.q5,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.dsc5,'BackgroundColor',[0.5020 0.5020 0.5020]);
end

data = getappdata(gcbf, 'metricdata');
data.boolean5 = aux;
setappdata(gcbf, 'metricdata', data);
```

```
if alguno_ON(gcbf,handles)==0
    set(handles.run,'enable','off');
else
    set(handles.run,'enable','on');
end


function booleanALG_Callback(hObject, eventdata, handles)
if get(hObject,'value')
    aux = 1;
    set(handles.archivoA,'enable','on');
    set(handles.archivoA,'BackgroundColor','white');
    set(handles.Palg,'BackgroundColor','white');
    set(handles.Qalg,'BackgroundColor','white');
    set(handles.DSCalg,'BackgroundColor','white');
else
    aux = 0;
    set(handles.archivoA,'enable','off');
    set(handles.archivoA,'BackgroundColor',[0.5020 0.5020 1.0000]);
    set(handles.Palg,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.Qalg,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.DSCalg,'BackgroundColor',[0.5020 0.5020 0.5020]);
end

data = getappdata(gcbf, 'metricdata');
data.booleanALG = aux;
setappdata(gcbf, 'metricdata', data);

if alguno_ON(gcbf,handles)==0
    set(handles.run,'enable','off');
else
    set(handles.run,'enable','on');
end



function booleanALG2_Callback(hObject, eventdata, handles)
if get(hObject,'value')
    aux = 1;
    set(handles.archivoA2,'enable','on');
    set(handles.archivoA2,'BackgroundColor','white');
    set(handles.Palg2,'BackgroundColor','white');
    set(handles.Qalg2,'BackgroundColor','white');
    set(handles.DSCalg2,'BackgroundColor','white');
else
    aux = 0;
    set(handles.archivoA2,'enable','off');
    set(handles.archivoA2,'BackgroundColor',[0.5020 0.5020 1.0000]);
    set(handles.Palg2,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.Qalg2,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.DSCalg2,'BackgroundColor',[0.5020 0.5020 0.5020]);
end

data = getappdata(gcbf, 'metricdata');
data.booleanALG2 = aux;
setappdata(gcbf, 'metricdata', data);

if alguno_ON(gcbf,handles)==0
    set(handles.run,'enable','off');
else
    set(handles.run,'enable','on');
end
```

```
function booleanALG3_Callback(hObject, eventdata, handles)
if get(hObject,'value')
    aux = 1;
    set(handles.archivoA3,'enable','on');
    set(handles.archivoA3,'BackgroundColor','white');
    set(handles.Palg3,'BackgroundColor','white');
    set(handles.Qalg3,'BackgroundColor','white');
    set(handles.DSCalg3,'BackgroundColor','white');
else
    aux = 0;
    set(handles.archivoA3,'enable','off');
    set(handles.archivoA3,'BackgroundColor',[0.5020 0.5020 1.0000]);
    set(handles.Palg3,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.Qalg3,'BackgroundColor',[0.5020 0.5020 0.5020]);
    set(handles.DSCalg3,'BackgroundColor',[0.5020 0.5020 0.5020]);
end

data = getappdata(gcbf, 'metricdata');
data.booleanALG3 = aux;
setappdata(gcbf, 'metricdata', data);

if alguno_ON(gcbf,handles)==0
    set(handles.run,'enable','off');
else
    set(handles.run,'enable','on');
end



%--------------------------------------------------------------------------------
%   BOTON PARA CAPTURAR LAS IMAGENES AL PRINCIPIO
%--------------------------------------------------------------------------------

% Al pulsar CAPTURAR se crea una pequeña lista con los nombres de los
% archivos que permitira visualizarlos si se desea:

function captura_Callback(hObject, eventdata, handles)
capturar_bis(gcbf,handles);




%--------------------------------------------------------------------------------
%   VISUALIZACION DE LAS IMAGENES CAPTURADAS
%--------------------------------------------------------------------------------

% Menu desplegable que se configura al pulsar CAPTURAR:

function ver_imags_CreateFcn(hObject, eventdata, handles)
if ispc
    %set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function ver_imags_Callback(hObject, eventdata, handles)

% Elemento seleccionado del menu desplegable:

popup = get(handles.ver_imags, 'Value');
```

```
% Visualizacion del correspondiente archivo de imagen:

visualiza_menu(gcbf,handles,popup,1);



%--------------------------------------------------------------------------
%   VISUALIZACION DE IMAGENES RESULTANTES DE LA CAPTURA Y SELECCION COLOR
%--------------------------------------------------------------------------

% Menu desplegable que contiene las imagenes binarias resultantes de todo
% el preprocesamiento, que son las entradas al algoritmo de validacion
% (nucleo del software diseñado)

function contornos_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function contornos_Callback(hObject, eventdata, handles)

% Elemento seleccionado del menu desplegable:

popup = get(handles.contornos, 'Value');

% Visualizacion del correspondiente archivo de imagen:

visualiza_menu(gcbf,handles,popup,0);



%--------------------------------------------------------------------------
%   VISUALIZACION DE LAS LAS MATRICES DE PESOS
%--------------------------------------------------------------------------

% Menu desplegable que contiene las matrices de pesos obtenidas en la
% primera y ultima iteracion del algoritmo

function mat_pesos_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function mat_pesos_Callback(hObject, eventdata, handles)

% Elemento seleccionado del menu desplegable:

popup = get(handles.mat_pesos, 'Value');

switch popup
    case 1
        % El primer elemento se refiere a la matriz inicial, incluida en el
        % archivo de salida 'W_inicial.bmp', que se visualiza con la
        % orden 'imshow':

        imshow('W_inicial.bmp');
```

```
    case 2
        % El segundo elemento se refiere a la matriz de pesos final,
        % incluida en el archivo de salida 'W_final.bmp', que se
        % visualiza con la orden 'imshow':

        imshow('W_final.bmp');
end



%--------------------------------------------------------------------------------
%   TEMPORIZACION
%--------------------------------------------------------------------------------

% Casillero en el que consta el tiempo maximo de ejecucion del programa

function timer_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function timer_Callback(hObject, eventdata, handles)



%--------------------------------------------------------------------------------
%   CREACION DE UN HISTORICO DE RESULTADOS EN LA MISMA VENTANA DE MATLAB
%--------------------------------------------------------------------------------

function historico_Callback(hObject, eventdata, handles)

% Ayuda en la linea de indicaciones de la interfaz:

set(handles.texto,'String','Opcion que permite presentar el historico de
                                resultados en la ventana de Matlab.');

%--------------------------------------------------------------------------------
%   OPCION PARA EVITAR ETAPA DE PROCESAMIENTO DE IMAGENES: PARTIR DE
%   IMAGENES YA BINARIAS. LOS CONTORNOS SIGNIFICATIVOS A CONSIDERAR YA
%   HAN SIDO SELECCIONADOS
%--------------------------------------------------------------------------------

function partirBIN_Callback(hObject, eventdata, handles)
set(handles.texto,'String','Opcion que permite partir desde imagenes ya
preprocesadas.');


%--------------------------------------------------------------------------------
%   BOTON DE RESET
%--------------------------------------------------------------------------------

function reset_Callback(hObject, eventdata, handles)

% Inicializacion de todos los valores:

inicia_valores(hObject, handles);
```

```
%--------------------------------------------------------------------------------
%   EJECUCION DEL PROGRAMA
%--------------------------------------------------------------------------------

% Al pulsar EJECUTAR:

function run_Callback(hObject, eventdata, handles)

% Ayuda en la linea de indicaciones de la interfaz:
set(handles.texto,'String','Leyendo datos ...');

data = getappdata(gcbf, 'metricdata');


% Opciones del menu desplegable de imagenes ya preprocesadas:

set(handles.contornos, 'String', {'seleccion1.bmp','seleccion2.bmp',
        'seleccion3.bmp','seleccion4.bmp','seleccion5.bmp','seleccion6.bmp',
        'seleccion7.bmp','seleccion8.bmp'});


% Creacion de un array de banderas que indican las opciones de archivos de
% entrada activadas:

aux(1,1) = get(handles.boolean1,'Value');
aux(1,2) = get(handles.boolean2,'Value');
aux(1,3) = get(handles.boolean3,'Value');
aux(1,4) = get(handles.boolean4,'Value');
aux(1,5) = get(handles.boolean5,'Value');
aux(1,6) = get(handles.booleanALG,'Value');
aux(1,7) = get(handles.booleanALG2,'Value');
aux(1,8) = get(handles.booleanALG3,'Value');


% Se cargan p_ini y q_ini. Creacion de arrays p, q y dsc:
[p,q,dsc] = solo_activos(gcbf, handles,aux);

% Lectura de la especificacion g(T=1):
g = str2double(get(handles.G,'String'));

% Captura de las imagenes de entrada:
B = capturar(gcbf,handles,aux);


% Modulo de preprocesamiento de imagenes:

set(handles.texto,'String','Seleccionar una region ...');
A = seleccion(gcbf, handles,B,aux);

%--------------------------------------------------------------------------------

% Linea de indicaciones de la interfaz:

set(handles.texto,'String','Iniciando el algoritmo ...');

% Calculo y almacenamiento en archivo de la primera matriz de pesos:

W = matriz_pesos(A,p,q,g);
imwrite(W,'W_inicial.bmp');
imshow(W),title('T inicial');
```

```
% Mostrar los resultados en los casilleros de salida:
mostrar_resultado(gcbf,handles,p,q,dsc,aux);

flag = 1;
cont = 0;

% Condicion de parada del algoritmo de validacion iterativo:
condicion = 0;

% bool_error = 1 si se especifico parada por error maximo ( = 0 eoc):

bool_error = get(handles.errorflag,'Value');


% bool_iter = 1 si se especifico parada por numero max de iteraciones ( = 0 eoc):

bool_iter = get(handles.Niterflag,'Value');


% Lectura del instante de comienzo de la parte iterativa para comprobar que
% no se superara el tiempo maximo de ejecucion del programa (se lee en segundos):

t_ini = clock;
t_fin = 60 * str2double(get(handles.timer,'String'));


% PARTE ITERATIVA:

while flag

   % Se muestra la iteracion actual en la linea de indicaciones:

   cadena = strcat('Iteracion: ',num2str(cont));
   set(handles.texto,'String',cadena);

   % Actualizacion del contador de iteraciones:

   cont = cont+1;

   % Actualizacion de parametros de la anterior iteracion (sirven para
   % detectar la convergencia del algoritmo):

   p_ant = p;
   q_ant = q;

   % Calculo de los parametros de calidad y del DSC:

   [p,q] = calculaP_pesos(A,W);
   dsc = calculaDSC(A,W);

   % Calculo de la nueva matriz de pesos:

   [W] = matriz_pesos(A,p,q,g);

   % Crear historico de resultados en archivo 'iteraciones.txt':

   historico(p,q,dsc,cont,aux);
```

```
   % Mostrar resultados en la ventana de MATLAB si se activo la correspondiente
   % opcion:

   if get(handles.historico,'Value')
       evolucion(p,q,dsc,cont,aux);
   end

   % Condicion de vencimiento de tiempo de ejecucion:
   condicion_timer = (etime(clock,t_ini) >= t_fin);


   % Condicion de convergencia absoluta:
   condicion_base = (p_ant == p)&(q_ant == q);


   % Se toma como condicion de parada la condicion de convergencia absoluta
   % si no se especifican opciones:

   if (bool_error) == 0 & (bool_iter == 0)
       condicion = condicion_base;
   end


   % Si se especifica opcion de parada por error maximo, la condicion final de
   % parada debe ser el OR de esta y de la de convergencia absoluta (por si la
   % convergencia absoluta se alcanza antes):

   if bool_error
       error = str2double(get(handles.error,'String'));
       condicion = ((abs(p_ant - p) <= error) & (abs(q_ant - q) <= error)) |
                                                              condicion_base;
   end

   % Si se especifica opcion de parada por numero maximo de iteraciones,
   % la condicion final de parada debe ser el OR de esta y de la de convergencia
   % absoluta (por si la convergencia absoluta se alcanza antes):

   if bool_iter
       iters = str2double(get(handles.Niter,'String'));
       condicion = (cont == iters) | condicion_base;
   end

   % Se deja de iterar si se cumple la condicion final de parada o si
   % transcurre el tiempo maximo de ejecucion:

   if condicion | condicion_timer
       flag = 0;
   end

   % Presentacion de los resultados en los casilleros de salida:
   mostrar_resultado(gcbf,handles,p,q,dsc,aux);

end

% Presentacion de los resultados en los casilleros de salida:

mostrar_resultado(gcbf,handles,p,q,dsc,aux);
```

```
% Visualizacion de la matriz de pesos final (ultima iteracion):

imshow(W),title('T final');
imwrite(W,'W_final.bmp');

% Se escribe el numero de iteraciones ejecutadas y el error maximo final entre
% ultimo y penultimo parametro:

err = error_maximo(p,q,p_ant,q_ant);
set(handles.errorFINAL,'String',err);
set(handles.NiterFINAL,'String',cont);

% Creacion y activacion del menu desplegable para las matrices de pesos inicial y
% final:

set(handles.mat_pesos, 'String', {'W inicial','W final'}, 'BackgroundColor',
                                                    'white','enable','on');

% Linea de indicaciones:
set(handles.texto,'String','Fin del programa y ficheros de salida creados. Pulsar
                                                      RESET para reinicio. ');

% Activacion del boton repetir:
set(handles.repetir,'enable','on');


% Activacion del menu desplegable de las imagenes resultantes del
% preprocesamiento:

set(handles.contornos,'BackgroundColor','white','enable','on');


%          ============================================================
%                              FIN DEL PROGRAMA
%          ============================================================




%--------------------------------------------------------------------------------
%   BOTON PARA REPETICION
%--------------------------------------------------------------------------------

% Es el mismo programa, pero se parte de las imagenes ya preprocesadas, por
% lo que se ahorra tiempo y se perimiten multiples pruebas sobre las
% imagenes:

function repetir_Callback(hObject, eventdata, handles)

% Linea de indicaciones de la interfaz:
set(handles.texto,'String','Iniciando el algoritmo ...');

% Creacion de un array de banderas que indican las opciones de archivos de
% entrada activadas:
aux(1,1) = get(handles.boolean1,'Value');
aux(1,2) = get(handles.boolean2,'Value');
aux(1,3) = get(handles.boolean3,'Value');
aux(1,4) = get(handles.boolean4,'Value');
aux(1,5) = get(handles.boolean5,'Value');
aux(1,6) = get(handles.booleanALG,'Value');
aux(1,7) = get(handles.booleanALG2,'Value');
aux(1,8) = get(handles.booleanALG3,'Value');
```

```
% Se capturan en array I las imagenes ya preprocesadas durante la primera
% ejecucion:

I = captura_repetir(gcbf,handles,aux);


% Se cargan p_ini y q_ini. Creacion de arrays p, q y dsc:
[p,q,dsc] = solo_activos(gcbf, handles,aux);

% Calculo de g(T=1) optima:
g = calculaG(A);
set(handles.G,'String',g);

% Calculo y almacenamiento en archivo de la primera matriz de pesos:

W = matriz_pesos(I,p,q,g);
imwrite(W,'W_inicial.bmp');
imshow(W),title('T inicial');

% Mostrar los resultados en los casilleros de salida:
mostrar_resultado(gcbf,handles,p,q,dsc,aux);

flag = 1;
cont = 0;

% Condicion de parada del algoritmo de validacion iterativo:
condicion = 0;

% bool_error = 1 si se especifico parada por error maximo ( = 0 eoc):
bool_error = get(handles.errorflag,'Value');

% bool_iter = 1 si se especifico parada por numero max de iteraciones (= 0 eoc):
bool_iter = get(handles.Niterflag,'Value');

% Lectura del instante de comienzo de la parte iterativa para comprobar que
% no se superara el tiempo maximo de ejecucion del programa (se lee en segundos):

t_ini = clock;
t_fin = 60 * str2double(get(handles.timer,'String'));


% PARTE ITERATIVA:

while flag

   % Se muestra la iteracion actual en la linea de indicaciones:
   cadena = strcat('Iteracion: ',num2str(cont));
   set(handles.texto,'String',cadena);

   % Actualizacion del contador de iteraciones:
   cont = cont+1;

   % Actualizacion de parametros de la anterior iteracion (sirven para
   % detectar la convergencia del algoritmo):
   p_ant = p;
   q_ant = q;


   % Calculo de los parametros de calidad y del DSC:
   [p,q] = calculaP_pesos(A,W);
   dsc = calculaDSC(A,W);
```

```
   % Calculo de la nueva matriz de pesos:
   [W] = matriz_pesos(A,p,q,g);

   % Crear historico de resultados en archivo 'iteraciones.txt':
   historico(p,q,dsc,cont,aux);

   % Mostrar resultados en la ventana de MATLAB si se activo la correspondiente
   % opcion:

   if get(handles.historico,'Value')
       evolucion(p,q,dsc,cont,aux);
   end

   % Condicion de vencimiento de tiempo de ejecucion:
   condicion_timer = (etime(clock,t_ini) >= t_fin);

   % Condicion de convergencia absoluta:
   condicion_base = (p_ant == p)&(q_ant == q);

   % Se toma como condicion de parada la condicion de convergencia absoluta
   % si no se especifican opciones:

   if (bool_error) == 0 & (bool_iter == 0)
       condicion = condicion_base;
   end

   % Si se especifica opcion de parada por error maximo, la condicion final de
   % parada debe ser el OR de esta y de la de convergencia absoluta (por si la
   % convergencia absoluta se alcanza antes):

   if bool_error
       error = str2double(get(handles.error,'String'));
       condicion = ((abs(p_ant - p) <= error) & (abs(q_ant - q) <= error)) |
                                                          condicion_base;
   end


   % Si se especifica opcion de parada por numero maximo de iteraciones,
   % la condicion final de parada debe ser el OR de esta y de la de convergencia
   % absoluta (por si la convergencia absoluta se alcanza antes):

   if bool_iter
       iters = str2double(get(handles.Niter,'String'));
       condicion = (cont == iters) | condicion_base;
   end

   % Se deja de iterar si se cumple la condicion final de parada o si
   % transcurre el tiempo maximo de ejecucion:

   if condicion | condicion_timer
       flag = 0;
   end

   % Presentacion de los resultados en los casilleros de salida:
   mostrar_resultado(gcbf,handles,p,q,dsc,aux);

end
```

```
% Presentacion de los resultados en los casilleros de salida:
mostrar_resultado(gcbf,handles,p,q,dsc,aux);

% Visualizacion de la matriz de pesos final (ultima iteracion):
imshow(W),title('T final');
imwrite(W,'W_final.bmp');

% Se escribe el numero de iteraciones ejecutadas y el error maximo final entre
% ultimo y penultimo parametro:
err = error_maximo(p,q,p_ant,q_ant);
set(handles.errorFINAL,'String',err);
set(handles.NiterFINAL,'String',cont);

% Creacion y activacion del menu desplegable para las matrices de pesos inicial y
% final:

set(handles.mat_pesos, 'String', {'W inicial','W final'},'BackgroundColor',
                                                    'white','enable','on');

% Linea de indicaciones:

set(handles.texto,'String','Fin del programa y ficheros de salida creados. Pulsar
RESET para reinicio. ');

% Activacion del boton repetir:

set(handles.repetir,'enable','on');

% Activacion del menu desplegable de las imagenes resultantes del
% preprocesamiento:

set(handles.contornos,'BackgroundColor','white','enable','on');
```

4.5.4 CÓDIGO DE LOS SUBPROGRAMAS

A continuación se detalla el código de todos los subprogramas correspondientes a esta última versión.

4.5.4.1 Subprogramas dedicados a la captura y procesamiento de imágenes

4.5.4.1.1 Capturar.m

Esta importante función se encarga de la lectura de los archivos de entrada y del almacenamiento de las imágenes en un array multidimensional, con el cual resultará más cómodo trabajar.

También se ejecutará el preprocesado adicional sobre las imágenes resultantes de los procesos automáticos de segmentación, si es que lo necesitan.

El código es el que se muestra seguidamente:

```
% Recopilacion de todas las imagenes en un array multidimensional B:

function B = capturar(gcbf,handles,aux)

% El indice 'm' indicara el sujeto al cual pertenece la imagen (entre 1 y 8):
m = 1;

for k=1:8
    if aux(1,k)

        % Captura de los nombres de los archivos:
        switch k
            case 1
                auxv = get(handles.archivo1, 'String');
            case 2
                auxv = get(handles.archivo2, 'String');
            case 3
                auxv = get(handles.archivo3, 'String');
            case 4
                auxv = get(handles.archivo4, 'String');
            case 5
                auxv = get(handles.archivo5, 'String');
            case 6
                auxv = get(handles.archivoA, 'String');
            case 7
                auxv = get(handles.archivoA2, 'String');
            case 8
                auxv = get(handles.archivoA3, 'String');

        end
```

```
        % Para los resultados de la segmentacion:

        if (k == 6)|(k == 7)|(k == 8) (*)

            aa = imread(auxv);

            % Se comprueba si es necesario el procesado adicional
            % (prepara_alg): Si el tamaño de la imagen no muestra
            % variaciones, no es necesario hacer operacion alguna
            % y simplemente se lee.

        % La variable 'tam' ya se creo al pasar por anteriores valores de 'k'.

            if size(aa)==tam
                B(:,:,:,m) = aa;
            else
                B(:,:,:,m) = prepara_alg(aa,tam);
            end


        % Para las imagenes de los expertos:

        else
            B(:,:,:,m) = imread(auxv);

            tam = size(B(:,:,:,m));

        end

        % Actualizacion del indice:
        m = m+1;

    end
end


return
```

Aspectos de consideración:

El programa admite colocar imágenes de expertos en los casilleros correspondientes a las de los programas de segmentación.

En cuanto a la acción opuesta, sería también conveniente evitar que el usuario se vea obligado a tomar un número máximo de imágenes resultantes de segmentación (permitiéndose así ocupar casilleros que podrían estar libres). En efecto, se puede optar por efectuar la comprobación (*) en cada uno de los sujetos (es decir, para todo k). Con esto sería posible rellenar los casilleros de entrada con archivos de cualquier tipo (expertos o algoritmos).

No obstante, la variable ‘tam’, que proporciona la información suficiente para distinguir las imágenes de uno y otro tipo, podría ser entonces motivo de error o confusión, por lo que habría que facilitar al programa el tamaño de las imágenes proporcionadas por los expertos, que conservan el original.

Finalmente no se ha decidido solucionar este pequeño inconveniente, dado que el número de casilleros es suficientemente grande. Aun así, no es probable que existan las alteraciones citadas si se llevan a cabo las acciones comentadas en 4.2.3 sobre los programas de segmentación utilizados, que motivarán el correcto funcionamiento del programa para un orden y tipo aleatorios de imágenes.

4.5.4.1.2 Prepara_alg.m

```
% FUNCIÓN QUE ADAPTA LA IMAGEN 'A' AL TAMAÑO TAM,
% ELIMINANDO EL MARCO BLANCO QUE LA RODEA, SI ES QUE EXISTE.

% alto: size(I,1)
% ancho: size(I,2)

function B = prepara_alg (A,tam)

i=1;
j=1;
flag=1;
x1=1;
y1=1;
x2=1;
y2=1;
margen=100;

% Detección de la esquina superior izquierda de la imagen
% rodeada por el marco blanco.

for i=1:size(A,1)
   for j=1:size(A,2)

        % Se tendrá en cuenta que la posición actual no pertenece al marco blanco
        % si el valor del píxel difiere de dicho color (255) en un margen mínimo
        % considerable.

        if ((abs(double(A(i,j,1))-255)>=margen)|(abs(double(A(i,j,2))-
                    255)>=margen)|(abs(double(A(i,j,3))-255)>=margen))&flag

            % Coordenadas esquina:
            x1=j;
            y1=i;

            % Fin del bucle:
            j=size(A,2);
            i=size(A,1);
            flag=0;
        end
    end
end

flag=1;
```

```
% Detección de la esquina inferior derecha de la imagen rodeada por el marco
% blanco.

for i=size(A,1):-1:1
   for j=size(A,2):-1:1
        % Se tendrá en cuenta que la posición actual no pertenece al
        % marco blanco si el valor del píxel difiere de dicho color
        % (255) en un margen mínimo considerable.

        if ((abs(double(A(i,j,1))-255)>=margen)|(abs(double(A(i,j,2))-
                    255)>=margen)|(abs(double(A(i,j,3))-255)>=margen))&flag

            % Coordenadas esquina:
            x2=j;
            y2=i;
            % Fin del bucle:
            j=1;
            i=1;
            flag=0;
        end
    end
end

% Dadas las coordenadas de las esquinas, se elimina el marco y se
% captura en 'B' la imagen en cuestión:

B = imcrop(A,[x1 y1 x2-x1 y2-y1]);

% Se reajusta el tamaño para hacerlo coincidir con el de las imágenes
% de los expertos, cuyo tamaño viene dado en el vactor 'tam':

B = imresize(B,[tam(1,1),tam(1,2)]);

return
```

4.5.4.1.3 Selección.m

Es uno de los subprogramas más importantes, dado que se encarga del preprocesamiento de las imágenes.

Su cometido es permitir al usuario seleccionar una ventana dentro de cada imagen para así disponer de imágenes de menor tamaño (o bien seleccionarlas por completo, que es como se ha procedido en la ejecución de los ejemplos). Además permite la selección del color con que ha sido tomada la decisión por parte de cada sujeto, dado que dicho color suele variar de unos a otros.

Finalmente, proporciona imágenes relativamente pequeñas (según la extensión de la región seleccionada) y binarias, cuyos píxeles valen 1 si su color correspondía con el del contorno seleccionado (o pertenecían a su interior) y 0 en caso contrario.

```
function C = seleccion(fig_handle, handles,B,aux)

% flag_bin = 1 si las imagenes de inicio ya son binarias, en cuyo caso
% unicamente se permite la opcion de seleccionar una subregion (no es
% necesaria la deteccion del color):

flag_BIN = get(handles.partirBIN,'Value');

% Seleccion de la subregion de la imagen RGB 'B':
% -----------------------------------------------
% La primera seleccion es manual (con el uso del raton):
[X,Y,N(:,:,:,1),RECT] = imcrop(B(:,:,:,1));

% Esta primera seleccion se almacena en un archivo
% del directorio de trabajo:
imwrite(N(:,:,:,1),'resultados/seleccion.bmp');

% Los valores de salida de la seleccion manual sirven para realizar,
% automaticamente y con identicas coordenadas, las restantes:

for k=2:size(B,4)
        N(:,:,:,k) = imcrop(B(:,:,:,k),RECT);
end

% Para reordenar numeros: Solo se creara seleccionI si I activado
% Con este bucle se almacenan en una tabla llamada 'num' los activados
% Ej: si solo se estan tratando los sujetos 3,4 y 7 se deben creas solamente
% los archivos: seleccion3.bmp, seleccion4.bmp y seleccion7.bmp, asi que
% se tendra: num = [3 4 7]:

i = 1;
for k=1:8
    if aux(1,k)
        num(1,i) = 48+k;
        i = i+1;
    end
end

if flag_BIN
    for k=1:size(B,4)
        % Asegurar que las imagenes contienen valores binarios:
        C(:,:,k) = double(N(:,:,:,k))/255;
    end
else
    % Especificacion del color a detectar en cada imagen:
    for k=1:size(B,4)
        % Linea de indicaciones:

        set(handles.texto,'String','Seleccionar region y pinchar sobre el color a
        detectar (INTRO)');

        % Para facilitar pinchar en el contorno:
        aux = imcrop(N(:,:,:,k));

        % Captura de las componentes RGB del pixel seleccionado:
        color(k,:) = impixel(aux);
    end
```

```
    % Mensaje para indicar que se esta detectando el color:
    set(handles.texto,'String','Procesando...');

    for i=1:size(B,4)
        % Deteccion del color:
        C(:,:,i) = detecta_color(N(:,:,:,i),color(i,:),20);

        % Se rellena el interior de los contornos:
        C(:,:,i) = bwfill(C(:,:,i),'holes');
    end
end

for i=1:size(B,4)
    % Se crean los archivos resultantes del preprocesamiento, llamados
    % seleccion(i).bmp

    word = strcat('resultados/seleccion',char(num(1,i)));
    file_out = strcat(word,'.bmp');
    imwrite(C(:,:,i),file_out);
end
return
```

4.5.4.1.4 Detecta_color.m

```
% Funcion que devuelve una imagen binaria resultante de la deteccion de un
% determinado color en la imagen de entrada M:

function A = detecta_color(M,c,margen)
i=1;
j=1;
for i=1:size(M,1)
   for j=1:size(M,2)
       % Se comprueba que el color del píxel (i,j) de la imagen es,
       % aproximadamente, el especificado:
       if proximo(M,c,margen,i,j,1) & proximo(M,c,margen,i,j,2) &
                                                        proximo(M,c,margen,i,j,3)
          A(i,j)=1;
       else
          A(i,j)=0;
       end
   end
end
return
```

4.5.4.1.5 Proximo.m

Este subprograma juega un papel importante en lo que se refiere a la detección del color en las imágenes. Una vez que se conoce el color a detectar, hay que buscarlo en la imagen y marcar aquellos píxeles cuyas componentes RGB sean suficientemente parecidas a dicho color. La selectividad de la búsqueda puede ser modificada con un determinado ***margen*** de error.

El programa se muestra a continuación.

```
% M: Imagen a color a considerar
% c: Vector que contiene las componentes del color a identificar en la imagen
% margen: Margen permitido entre las componentes RGB
% i,j: Coordenadas del pixel actual
% k: Componente que se esta comprobando (k=1: R, k=2: G, k=3: B).


function a = proximo(M,c,margen,i,j,k)

if (abs(double(M(i,j,k))-double(c(k)))<=margen)
  a=1;
else
  a=0;
end

% Se devuelve 1 si las componentes del pixel son suficientemente parecidas
% al color buscado, y 0 en caso contrario.

return
```

4.5.4.2 Subprogramas correspondientes al algoritmo de validación

Los subprogramas y operaciones correspondientes al algoritmo de validación ya fueron analizados en el apartado 4.3. El código de cada uno de ellos se muestra a continuación:

4.5.4.2.1 CalculaG.m

```
% Funcion que calcula el factor de ponderacion g(T=1)

function g = calculaG(A)

unos = 0;
ceros = 0;
suma = 0;

for k=1:size(A,3)

    for i=1:size(A,1)
        for j=1:size(A,2)

            % Calculo del numero de pixeles de valor 1 en la imagen k:
            if A(i,j,k)==1
                unos = unos +1;
            end

            % Calculo del numero de pixeles de valor 0 en la imagen k:
            if A(i,j,k)==0
                ceros = ceros +1;
            end
```

```
            % Porcion de la imagen total que ocupa la region significativa de la
            % imagen k:

            proporcion(1,k) = unos/(unos + ceros);

        end
    end
    unos = 0;
    ceros = 0;

end

% Para g(T=1) se escoge la media de las relaciones unos-ceros de todas las
% imagenes:

for k=1:size(A,3)
    suma = suma + proporcion(1,k);
end

% Valor de g(T=1):
g = suma/size(A,3);

return
```

4.5.4.2.2 Matriz_pesos.m

```
% Programa que calcula la matriz de pesos W a partir de las imágenes
% binarias de entrada, la ponderación g(T) y los parámetros de calidad


function W = matriz_pesos(A,p,q,g)

alfa=1;
beta=1;

for i=1:size(A,1)
    for j=1:size(A,2)

        for k=1:size(A,3)

             % A(i,j,k) corresponde al píxel (i,j) de la imagen del
             % experto o algoritmo j-ésimo.

            alfa = alfa*(p(1,k)*(A(i,j,k)==1)+(1-p(1,k))*(A(i,j,k)==0) );
            beta = beta*(q(1,k)*(A(i,j,k)==0)+(1-q(1,k))*(A(i,j,k)==1) );
        end

        % Expresión 4.3.2.1

        W(i,j) = (g * alfa) / ( (g*alfa) + ((1-g)*beta) );

        alfa = 1;
        beta = 1;
    end
end

return
```

4.5.4.2.3 CalculaP_pesos

```
% Programa que obtiene el valor de los parámetros de calidad para cada experto
% y/o algoritmo a partir de las imágenes binarias de entrada y de la matriz de
% pesos W


function [p,q] = calculaP_pesos(A,W)

% Sumatorios

sum_p1=0;
sum_p2=0;

sum_q1=0;
sum_q2=0;

% Para cada experto/algoritmo k:
for k=1:size(A,3)
    for i=1:size(A,1)
        for j=1:size(A,2)

            sum_p1 = sum_p1 + W(i,j)*(A(i,j,k)==1);
            sum_p2 = sum_p2 + W(i,j)*(A(i,j,k)==0);

            sum_q1 = sum_q1 + (1-W(i,j))*(A(i,j,k)==0);
            sum_q2 = sum_q2 + (1-W(i,j))*(A(i,j,k)==1);
        end
    end


% Expresión 4.3.2.2

    p(1,k) = sum_p1 / ( sum_p1 + sum_p2 );

    q(1,k) = sum_q1 / ( sum_q1 + sum_q2 );

    sum_p1=0;
    sum_p2=0;

    sum_q1=0;
    sum_q2=0;
end

return
```

4.5.4.2.4 CalculaDSC.m

```
% Programa que calcula el “Dice Similarity Coeficient” a partir de las
% imágenes binarias de entrada y la matriz de pesos.

function DSC = calculaDSC(A,W)


% Para cada experto o algoritmo

for k=1:size(A,3)

    inter = 0;
    cont_A = 0;
    cont_W = 0;

    % Se recorre cada imagen binaria

    for i=1:size(W,1)
        for j=1:size(W,2)

            % Cálculo de la intersección:

            if (A(i,j,k)==1 & W(i,j)>=0.5)
                inter = inter + 1;
            end

            % Área de A:

            if A(i,j,k)==1
                cont_A = cont_A + 1;
            end

            % Área de B:

            if W(i,j)>=0.5
                cont_W = cont_W + 1;
            end

        end
    end


    % Expresión 4.3.2.3:

    DSC(1,k) = 2*inter/(cont_A + cont_W);

end
```

4.5.4.2.5 Error_maximo.m

Cuando no se establece ninguna condicion de parada, el programa realizará iteraciones hasta alcanzarse la convergencia absoluta (error 0) o hasta el fin del tiempo máximo de ejecución.

Si se elige un determinado número de iteraciones o se fija un error máximo, el casillero que muestra el error final indicará cuál es la máxima diferencia entre el valor, en la última iteración, de un determinado parámetro de calidad y éste mismo en la iteración previa.

Dicha diferencia debe ser menor o igual que el error especificado (si fue el error el factor determinante configurado para la parada del algoritmo).

El código de la función que calcula esta diferencia máxima a presentar en el correspondiente casillero es:

```
function err = error_maximo(p,q,p_ant,q_ant)

max1 = 0;
max2 = 0;

for i=1:size(p,2)

    % Calculo del maximo error en los parametros P:
    if abs(p(1,i)-p_ant(1,i))>=max1
        max1 = abs(p(1,i)-p_ant(1,i));
    end



    % Calculo del maximo error en los parametros Q:

    if abs(q(1,i)-q_ant(1,i))>=max2
        max2 = abs(p(1,i)-p_ant(1,i));
    end
end


% La diferencia maxima total se escogera como la mayor de las dos anteriores:

if max1 >= max2
    err = max1;
else
    err = max2;
end

return
```

4.5.4.3 Subprogramas que constituyen el entorno gráfico

4.5.4.3.1 Inicia_valores.m

Este subprograma se ejecuta cuando el usuario pulsa RESET. Se encarga de reiniciar todos y cada uno de los valores de la interfaz, ya sea el contenido numérico de los casilleros o bien se trate de la configuración inicial de cada uno de los controles (activación, color, etc.).

El código debidamente comentado se escribe a continuación:

```
% INICIALIZACION DE TODOS LOS VALORES (RESET):

function inicia_valores(fig_handle, handles)

    % Se muestra una imagen de presentacion:
    imshow(imread('presentacion.bmp'));

    % Linea de indicaciones bajo la region de la imagen:

    set(handles.texto,'String','Seleccionar archivos y parametros (Pulsar
EJECUTAR) ...');

    % Se comienza con los botones EJECUTAR y REPETIR desactivados (dado que
    % inicialmente no se activa ningun sujeto:

    set(handles.run,'enable','off');
    set(handles.repetir,'enable','off');

    % Configuracion del valor inicial por defecto de los parametros de
    % calidad y de g(T=1):

    set(handles.Pini,'String',0.9);
    set(handles.Qini,'String',0.9);
    set(handles.G,'String',0.5);


    % Se almacenan en variables:

    p_ini = str2double(get(handles.Pini,'String'));
    q_ini = str2double(get(handles.Qini,'String'));


    % Se desactivan las opciones de introduccion de imagenes:
    set(handles.boolean1, 'Value', 0);
    set(handles.boolean2, 'Value', 0);
    set(handles.boolean3, 'Value', 0);
    set(handles.boolean4, 'Value', 0);
    set(handles.boolean5, 'Value', 0);
    set(handles.booleanALG, 'Value', 0);
    set(handles.booleanALG2, 'Value', 0);
    set(handles.booleanALG3, 'Value', 0);
```

```
% Todos los casilleros de nombres de archivo de entrada son
% desactivados (y se les establece color del fondo):

set(handles.archivo1,'BackgroundColor',[0.5020 0.5020 1.000],'enable','off');
set(handles.archivo2,'BackgroundColor',[0.5020 0.5020 1.000],'enable','off');
set(handles.archivo3,'BackgroundColor',[0.5020 0.5020 1.000],'enable','off');
set(handles.archivo4,'BackgroundColor',[0.5020 0.5020 1.000],'enable','off');
set(handles.archivo5,'BackgroundColor',[0.5020 0.5020 1.000],'enable','off');
set(handles.archivoA,'BackgroundColor',[0.5020 0.5020 1.000],'enable','off');
set(handles.archivoA2,'BackgroundColor',[0.5020 0.5020 1.00],'enable','off');
set(handles.archivoA3,'BackgroundColor',[0.5020 0.5020 1.00],'enable','off');

% Se desactiva y se le establece el color del fondo al menu desplegable de
% visualizacion de las imagenes capturadas (antes del preprocesamiento):

set(handles.ver_imags,'BackgroundColor',[0.5020 0.502 0.502],'enable','off');

% Se establece un valor por defecto para las condiciones de parada:

set(handles.errorflag,'Value',1);
set(handles.error,'String',0.000001,'BackgroundColor','white','enable','on');

set(handles.Niterflag,'Value',0);
set(handles.Niter,'String',10);

% Opcion para obtener el historico de resultados desactivada:
set(handles.historico,'Value',0);

% Opcion de comienzo a partir de imagenes binarias desactivada:
set(handles.partirBIN,'Value',0);

% Establecimiento del valor por defecto del tiempo maximo de ejecucion:
set(handles.timer,'String',20);


% CASILLEROS DE SALIDA:
% =====================

% Establecimiento del valor inicial del parametro P para cada sujeto (y color
% de fondo):

set(handles.Palg,'String',p_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.Palg2,'String',p_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.Palg3,'String',p_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.p1,'String',p_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.p2,'String',p_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.p3,'String',p_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.p4,'String',p_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.p5,'String',p_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);


% Establecimiento del valor inicial del parametro Q para cada sujeto (y color
% de fondo):

set(handles.Qalg,'String',q_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.Qalg2,'String',q_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.Qalg3,'String',q_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.q1,'String',q_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.q2,'String',q_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.q3,'String',q_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.q4,'String',q_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.q5,'String',q_ini,'BackgroundColor',[0.5020 0.5020 0.5020]);
```

```
% Establecimiento del valor inicial del parametro DSC para cada sujeto (y
% color de fondo):

set(handles.DSCalg,'String',0.9,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.DSCalg2,'String',0.9,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.DSCalg3,'String',0.9,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.dsc1,'String',0.9,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.dsc2,'String',0.9,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.dsc3,'String',0.9,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.dsc4,'String',0.9,'BackgroundColor',[0.5020 0.5020 0.5020]);
set(handles.dsc5,'String',0.9,'BackgroundColor',[0.5020 0.5020 0.5020]);


% Inicializacion del valor para el error y numero de iteraciones
% resultantes:

set(handles.NiterFINAL,'String',0,'ForegroundColor','black');
set(handles.errorFINAL,'String',0,'ForegroundColor','black');


% Al principio se desactiva el menu desplegable para imagenes resultantes del
% preprocesamiento (dado que este aun no se ha efectuado). Se les establece
% el color del fondo:

set(handles.contornos,'BackgroundColor',[0.5020 0.502 0.502],'enable','off');
set(handles.mat_pesos,'BackgroundColor',[0.5020 0.502 0.502],'enable','off');
```

4.5.4.3.2 Solo_activos.m

Esta función crea tres arrays. Uno de ellos contiene únicamente los valores de las probabilidades P de cada experto o programa de validación que haya sido convenientemente activado. Otro contiene los valores de Q, y el último los del DSC. De esta forma resulta muy cómodo el flujo de datos entre los diversos subprogramas que constituyen el algoritmo de validación.

Estos arrays contendrán los valores iniciales de los parámetros, bien sean los establecidos por el usuario, o bien los valores por defecto.

```
function [p,q,dsc] = solo_activos(fig_handle,handles,aux)

% Arrays que incluyen los valores iniciales de P, Q y DSC para los 8
% sujetos:

for i=1:8
    pp(1,i) = str2double(get(handles.Pini,'String'));
end

for i=1:8
    qq(1,i) = str2double(get(handles.Qini,'String'));
end

for i=1:8
    dssc(1,i) = str2double(get(handles.dsc1,'String'));
end
```

```
m = 1;

% Se rellenan los arrays de salida con tan solo los datos relativos a los
% sujetos que se hallan activados, es decir, los que se estan teniendo en
% cuenta en la presente ejecucion:


% aux(1,k) = 1 si sujeto k-ésimo activado, 0 eoc.

for k=1:8
    if aux(1,k)
        p(1,m) = pp(1,k);
        q(1,m) = qq(1,k);
        dsc(1,m) = dssc(1,k);
        m = m+1;
    end
end
```

4.5.4.3.3 Capturar_bis.m

Cuando se pulsa CAPTURAR se crea un menú desplegable que contiene los nombres de las imágenes de entrada, para que así puedan visualizarse cuando el usuario lo desee, con el objetivo de comprobar que son las que efectivamente se buscan oque, en efecto, guardan idénticas propiedades, que es una de las condiciones del programa.

El código del subprograma es el siguiente:

```
% Creacion de un menu desplegable con los nombres de las imagenes al pulsar
% CAPTURAR:

function capturar_bis(gcbf,handles)

% aux(1.j) vale 1 si esta activada una imagen para el sujeto j.

aux(1,1) = get(handles.boolean1,'Value');
aux(1,2) = get(handles.boolean2,'Value');
aux(1,3) = get(handles.boolean3,'Value');
aux(1,4) = get(handles.boolean4,'Value');
aux(1,5) = get(handles.boolean5,'Value');
aux(1,6) = get(handles.booleanALG,'Value');
aux(1,7) = get(handles.booleanALG2,'Value');
aux(1,8) = get(handles.booleanALG3,'Value');

% Valores por defecto:
cad1 = '--';
cad2 = '--';
cad3 = '--';
cad4 = '--';
cad5 = '--';
cad6 = '--';
cad7 = '--';
cad8 = '--';
```

```
% Lectura de nombres de archivo:
for k=1:8
    if aux(1,k)
        switch k
            case 1
                auxv = get(handles.archivo1, 'String');
                cad1 = auxv;
            case 2
                auxv = get(handles.archivo2, 'String');
                cad2 = auxv;
            case 3
                auxv = get(handles.archivo3, 'String');
                cad3 = auxv;
            case 4
                auxv = get(handles.archivo4, 'String');
                cad4 = auxv;
            case 5
                auxv = get(handles.archivo5, 'String');
                cad5 = auxv;
            case 6
                auxv = get(handles.archivoA, 'String');
                cad6 = auxv;
            case 7
                auxv = get(handles.archivoA2, 'String');
                cad7 = auxv;
            case 8
                auxv = get(handles.archivoA3, 'String');
                cad8 = auxv;
        end
    end
end


% Creacion del menu con los nombres de los archivos o el nombre por defecto:

set(handles.ver_imags, 'String', {cad1,cad2, cad3, cad4,cad5,cad6,cad7,cad8},
                                  'BackgroundColor','white','enable','on');
```

4.5.4.3.4 Alguno_ON.m

Este sencillo programa permite conocer si se halla activada alguna de las ocho opciones de entrada de archivos, para así permitir al usuario pulsar EJECUTAR (no tiene sentido activar el pulsador si no se seleccionan archivos; es más, daría lugar a errores).

El código se muestra a continuación:

```
% Devuelve el numero de opciones de entrada de archivos que estan activadas

function nON = alguno_ON(gcbf,handles)

aux(1,1) = get(handles.boolean1,'Value');
aux(1,2) = get(handles.boolean2,'Value');
aux(1,3) = get(handles.boolean3,'Value');
aux(1,4) = get(handles.boolean4,'Value');
aux(1,5) = get(handles.boolean5,'Value');
aux(1,6) = get(handles.booleanALG,'Value');
aux(1,7) = get(handles.booleanALG2,'Value');
aux(1,8) = get(handles.booleanALG3,'Value');


nON=0;

for i=1:size(aux,2)
    if aux(1,i)
        nON = nON+1;
    end
end

return
```

4.5.4.3.5 Evolucion.m

Esta función permite al usuario (si activa la correspondiente petición) visualizar en la ventana de comandos de MATLAB el valor de los parámetros P, Q y DSC en todas las iteraciones efectuadas durante la ejecución del programa, por si se ve interesado en observar la trayectoria seguida por dichos valores hasta que se alcanza la convergencia o se verifica alguna de las condiciones de parada configuaradas antes de iniciar el algoritmo de validación.

```
function evolucion(p,q,dsc,cont,aux)

if cont == 1

    % Diseño de un encabezado en la primera iteracion:
    disp(sprintf('\n HISTORICO DE RESULTADOS\n ======================\n'));
    disp(sprintf('        Experto 1  Experto 2  Experto 3  Experto 4  Experto 5
                                    Algoritmo1 Algoritmo2 Algoritmo3'));
    disp(sprintf('-------------------------------------------------------------'));
end
```

```
% Presentacion de los valores:

disp(sprintf(' P(%d):  ',cont));

m = 1;
for i=1:8
    if aux(1,i)
        disp(sprintf('\b%f   ',p(1,m)));
        m = m+1;
    else
        disp(sprintf('\b--------   '));
    end
end

m = 1;

disp(sprintf(' Q(%d):  ',cont));
for i=1:8
    if aux(1,i)
        disp(sprintf('\b%f   ',q(1,m)));
        m = m+1;
    else
        disp(sprintf('\b--------   '));
    end
end

m = 1;
disp(sprintf(' D(%d):  ',cont));
for i=1:8
    if aux(1,i)
        disp(sprintf('\b%f   ',dsc(1,m)));
        m = m+1;
    else
        disp(sprintf('\b--------   '));
    end
end
disp(sprintf('-----------------------------------------------------------------'));
```

4.5.4.3.6 Historico.m

Es similar al anterior, pero crea un archivo en el directorio de trabajo que contiene la misma información, es decir, los valores de los parámetros en cada una de las iteraciones.

Como entradas, se le deben pasar los arrays p, q y dsc, así como la iteración actual en la que se halla el programa (*cont*).

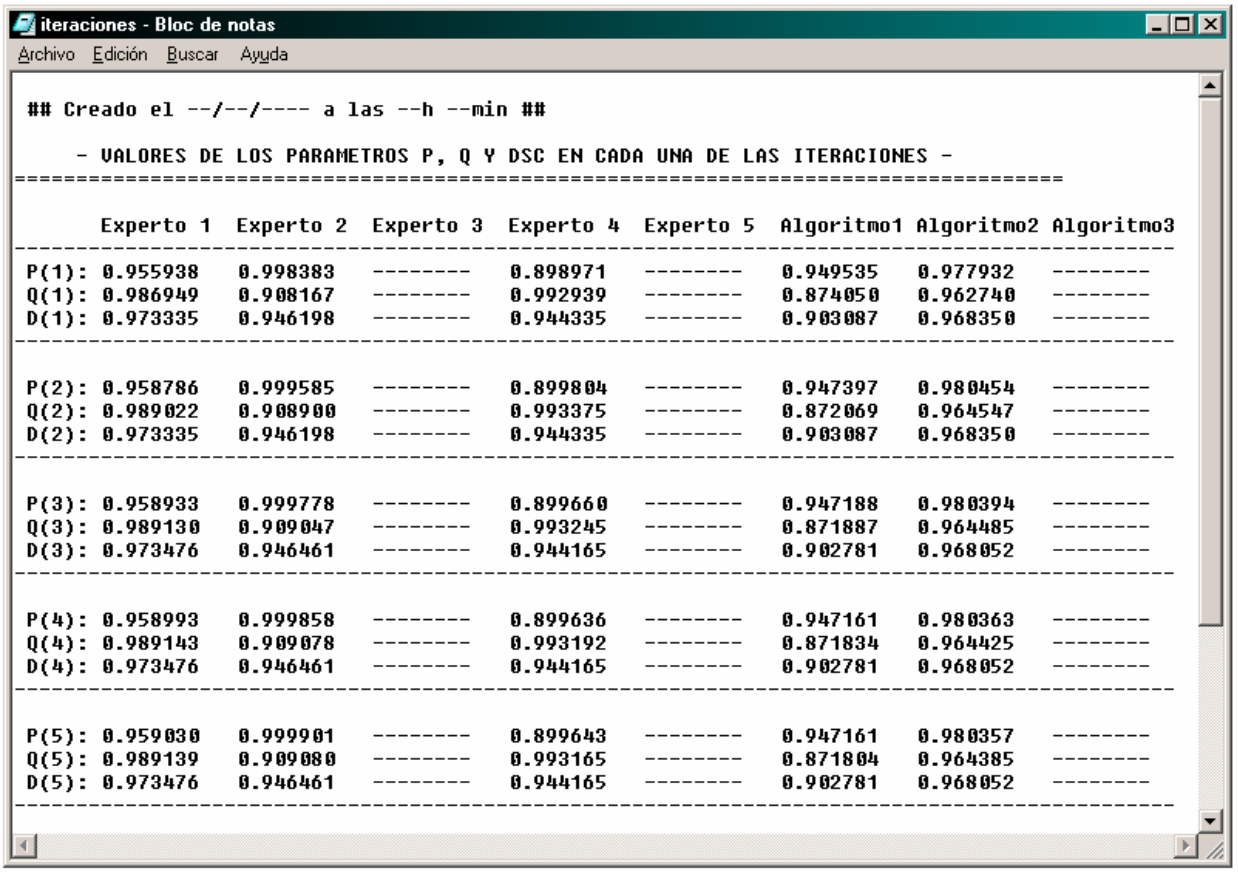
El código es el siguiente:

```
% CREACION DE UN ARCHIVO EN EL QUE CONSTAN LOS RESULTADOS DE CADA
% ITERACION:

function historico(p,q,dsc,cont,aux)

% Se aprovecha la primera iteracion para:
%   1 -  Leer el instante en que se produce
%   2 -  Crear el correspondiente archivo en escritura, activando la opcion de
%           truncar para escribir sobre anteriores tablas
%   3 -  Diseñar un encabezado que permita una comoda lectura de los resultados:

if cont == 1

   % Instante de comienzo:
   t = clock;
   dia = t(1,3);
   mes = t(1,2);
   ano = t(1,1);
   hor = t(1,4);
   min = t(1,5);

   % Creacion del archivo:
   mem = fopen('iteraciones.txt','w+t');  % opcion 'w+t' para comenzar de nuevo.


   % Diseño del encabezado:

   fprintf(mem,'\n ## Creado el %d/%d/%d a las %dh %dmin ##\n',dia,mes,ano,hor,
                                                                        min);

   fprintf(mem,'\n     - VALORES DE LOS PARAMETROS P, Q Y DSC EN CADA UNA DE LAS
                                                          ITERACIONES -\n');

   fprintf(mem,'=============================================================');

   fprintf(mem,'\n\n       Experto 1  Experto 2  Experto 3  Experto 4  Experto 5
                                        Algoritmo1 Algoritmo2 Algoritmo3');

   fprintf(mem,'\n-------------------------------------------------------------');


else
   % Si no es la primera iteracion, se iran añadiendo lineas al archivo
   % (opcion 'at' para continuar donde se dejo):

   mem = fopen('iteraciones.txt','at');
end
```

```
% Escritura de cada parametro en una linea nueva:

m = 1;
fprintf(mem,'\n P(%d): ',cont);

for i=1:8
    if aux(1,i)
        fprintf(mem,'%f   ',p(1,m));
        m = m+1;
    else
        fprintf(mem,'--------   ');
    end
end

m = 1;
fprintf(mem,'\n Q(%d): ',cont);

for i=1:8
    if aux(1,i)
        fprintf(mem,'%f   ',q(1,m));
        m = m+1;
    else
        fprintf(mem,'--------   ');
    end
end

m = 1;
fprintf(mem,'\n D(%d): ',cont);
for i=1:8
    if aux(1,i)
        fprintf(mem,'%f   ',dsc(1,m));
        m = m+1;
    else
        fprintf(mem,'--------   ');
    end
end

% Separador de lineas:
fprintf(mem,'\n-----------------------------------------------------------------\n');

% Se cierra el archivo:
fclose(mem);
```

EJEMPLO:

La apariencia del archivo ***iteraciones.txt*** creado durante la ejecución del programa sería la siguiente (para unas imágenes cualesquiera):



iteraciones - Bloc de notas

Archivo Edición Buscar Ayuda

```
## Creado el --/--/---- a las --h --min ##

    - VALORES DE LOS PARAMETROS P, Q Y DSC EN CADA UNA DE LAS ITERACIONES -
================================================================================

      Experto 1  Experto 2  Experto 3  Experto 4  Experto 5  Algoritmo1 Algoritmo2 Algoritmo3
--------------------------------------------------------------------------------------------
P(1): 0.955938   0.998383   --------   0.898971   --------   0.949535   0.977932   --------
Q(1): 0.986949   0.908167   --------   0.992939   --------   0.874050   0.962740   --------
D(1): 0.973335   0.946198   --------   0.944335   --------   0.903087   0.968350   --------
--------------------------------------------------------------------------------------------

P(2): 0.958786   0.999585   --------   0.899804   --------   0.947397   0.980454   --------
Q(2): 0.989022   0.908900   --------   0.993375   --------   0.872069   0.964547   --------
D(2): 0.973335   0.946198   --------   0.944335   --------   0.903087   0.968350   --------
--------------------------------------------------------------------------------------------

P(3): 0.958933   0.999778   --------   0.899660   --------   0.947188   0.980394   --------
Q(3): 0.989130   0.909047   --------   0.993245   --------   0.871887   0.964485   --------
D(3): 0.973476   0.946461   --------   0.944165   --------   0.902781   0.968052   --------
--------------------------------------------------------------------------------------------

P(4): 0.958993   0.999858   --------   0.899636   --------   0.947161   0.980363   --------
Q(4): 0.989143   0.909078   --------   0.993192   --------   0.871834   0.964425   --------
D(4): 0.973476   0.946461   --------   0.944165   --------   0.902781   0.968052   --------
--------------------------------------------------------------------------------------------

P(5): 0.959030   0.999901   --------   0.899643   --------   0.947161   0.980357   --------
Q(5): 0.989139   0.909080   --------   0.993165   --------   0.871804   0.964385   --------
D(5): 0.973476   0.946461   --------   0.944165   --------   0.902781   0.968052   --------
--------------------------------------------------------------------------------------------
```

4.5.4.3.7 Mostrar_resultado.m

Este programa se encarga de presentar los resultados de la ejecución en los correspondientes casilleros de la interfaz.

El código es el siguiente:

```
% PRESENTACION DE RESULTADOS:

% Se copia el contenido de los arrays p,q y dsc en los correspondientes
% casilleros. Si no estan activados, se escribe '--'.

function mostrar_resultado(fig_handle,handles,p,q,dsc,aux)

m = 1;
for k=1:8
    if aux(1,k)
        switch k
            case 1
                set(handles.p1,'String',p(1,m));
                set(handles.q1,'String',q(1,m));
                set(handles.dsc1,'String',dsc(1,m));
            case 2
                set(handles.p2,'String',p(1,m));
                set(handles.q2,'String',q(1,m));
                set(handles.dsc2,'String',dsc(1,m));
            case 3
                set(handles.p3,'String',p(1,m));
                set(handles.q3,'String',q(1,m));
                set(handles.dsc3,'String',dsc(1,m));
            case 4
                set(handles.p4,'String',p(1,m));
                set(handles.q4,'String',q(1,m));
                set(handles.dsc4,'String',dsc(1,m));
            case 5
                set(handles.p5,'String',p(1,m));
                set(handles.q5,'String',q(1,m));
                set(handles.dsc5,'String',dsc(1,m));
            case 6
                set(handles.Palg,'String',p(1,m));
                set(handles.Qalg,'String',q(1,m));
                set(handles.DSCalg,'String',dsc(1,m));
            case 7
                set(handles.Palg2,'String',p(1,m));
                set(handles.Qalg2,'String',q(1,m));
                set(handles.DSCalg2,'String',dsc(1,m));
            case 8
                set(handles.Palg3,'String',p(1,m));
                set(handles.Qalg3,'String',q(1,m));
                set(handles.DSCalg3,'String',dsc(1,m));
        end
        m = m+1;
```

```
   else
        switch k
            case 1
                set(handles.p1,'String','--');
                set(handles.q1,'String','--');
                set(handles.dsc1,'String','--');
            case 2
                set(handles.p2,'String','--');
                set(handles.q2,'String','--');
                set(handles.dsc2,'String','--');
            case 3
                set(handles.p3,'String','--');
                set(handles.q3,'String','--');
                set(handles.dsc3,'String','--');
            case 4
                set(handles.p4,'String','--');
                set(handles.q4,'String','--');
                set(handles.dsc4,'String','--');
            case 5
                set(handles.p5,'String','--');
                set(handles.q5,'String','--');
                set(handles.dsc5,'String','--');
            case 6
                set(handles.Palg,'String','--');
                set(handles.Qalg,'String','--');
                set(handles.DSCalg,'String','--');
            case 7
                set(handles.Palg2,'String','--');
                set(handles.Qalg2,'String','--');
                set(handles.DSCalg2,'String','--');
            case 8
                set(handles.Palg3,'String','--');
                set(handles.Qalg3,'String','--');
                set(handles.DSCalg3,'String','--');

        end
    end
end
```

4.5.4.3.8 Captura repetir.m

Esta función es llamada al pulsarse REPETIR, y es cuando se deben capturar las imágenes resultantes del preprocesamiento de una ejecución anterior del programa. El hecho de almacenar estos resultados en archivos de imagen hace posible el repetir la ejecución desde un punto más avanzado, ahorrándose así una nueva selección de regiones y colores, pudiéndose probar el mismo ejemplo para diferentes configuraciones.

El código es el siguiente:

```
% Recoge los datos de los contornos para ahorrar nueva seleccion:

% Se almacenan en un array A las imagenes en blanco y negro que son el
% resultado de la detección del color en las imágenes de entrada correspondientes
% a la ejecución anterior del mismo ejemplo. Estas se almacenaron en archivos
% tipo BMP, y contienen píxeles de valor 0 ó 255. Para operar con ellas es
% preciso realizar la conversión a imágenes binarias, dividiendo el valor de cada
% píxel por 255.

function A = captura_repetir(gcbf,handles,aux)

m = 1;

for k=1:8
       if aux(1,k)
              switch k
                     case 1
                        imag = imread('seleccion1.bmp');
                        A(:,:,m) = double(imag)/255;
                     case 2
                        imag = imread('seleccion2.bmp');
                        A(:,:,m) = double(imag)/255;
                     case 3
                        imag = imread('seleccion3.bmp');
                        A(:,:,m) = double(imag)/255;
                     case 4
                        imag = imread('seleccion4.bmp');
                        A(:,:,m) = double(imag)/255;
                     case 5
                        imag = imread('seleccion5.bmp');
                        A(:,:,m) = double(imag)/255;
                     case 6
                        imag = imread('seleccion6.bmp');
                        A(:,:,m) = double(imag)/255;
                     case 7
                        imag = imread('seleccion7.bmp');
                        A(:,:,m) = double(imag)/255;
                     case 8
                        imag = imread('seleccion8.bmp');
                        A(:,:,m) = double(imag)/255;
               end

               m = m+1;
       end
```

```
end

return
```

4.5.4.3.9 Visualiza_menu.m

Una vez creados los menús desplegables, esta función hace que, al pinchar sobre un determinado elemento de dicho menú (dado en el argumento *popup*, que será un número entre 1 y el número de elementos del menú), se proceda con la representación de la imagen seleccionada. Con la variable *opcion* se distingue entre el menu de imágenes capturadas al inicio (con el botón CAPTURAR) y el de contornos, que contiene las imágenes binarias obtenidas tras el preprocesamiento.

```
function visualiza_menu(gcbf,handles,popup,opcion)

% opcion = 1 cuando sea para los para las de captura inicial
% opcion = 0 para las resultantes de 'seleccion'

tam = [0 0 0];

switch popup
    case 1

        % Si opcion del primer archivo activada:

        if get(handles.boolean1,'Value')

            % Captura inicial de imagenes:
            if opcion

                % Captura del nombre del primer archivo:
                auxv = get(handles.archivo1, 'String');

                % Tamaño de la imagen contenida:
                tam = size(imread(auxv));
            else
                % Visualizacion de las imagenes resultantes del
                % preprocesamiento:
                auxv = 'seleccion1.bmp';
            end

        else
            % Si se ha pinchado sobre un archivo no existente se representa
            % una imagen por defecto:
            auxv = 'simbolo.bmp';
       end

    case 2
        if get(handles.boolean2,'Value')
            if opcion
                auxv = get(handles.archivo2, 'String');
                tam = size(imread(auxv));
            else
                auxv = 'seleccion2.bmp';

            end
```

```
        else
            auxv = 'simbolo.bmp';
        end

    case 3
        if get(handles.boolean3,'Value')
            if opcion
                auxv = get(handles.archivo3, 'String');
                tam = size(imread(auxv));
            else
                auxv = 'seleccion3.bmp';
            end
        else
            auxv = 'simbolo.bmp';
        end

    case 4
        if get(handles.boolean4,'Value')
            if opcion
                auxv = get(handles.archivo4, 'String');
                tam = size(imread(auxv));
            else
                auxv = 'seleccion4.bmp';
            end
        else
            auxv = 'simbolo.bmp';
        end

    case 5
        if get(handles.boolean5,'Value')
            if opcion
                auxv = get(handles.archivo5, 'String');
                tam = size(imread(auxv));
            else
                auxv = 'seleccion5.bmp';
            end
        else
            auxv = 'simbolo.bmp';
        end

    case 6
        if get(handles.booleanALG,'Value')
            if opcion
                auxv = get(handles.archivoA, 'String');
                tam = size(imread(auxv));
            else
                auxv = 'seleccion6.bmp';
            end
        else
            auxv = 'simbolo.bmp';
        end


    case 7
        if get(handles.booleanALG2,'Value')
            if opcion
                auxv = get(handles.archivoA2, 'String');
                tam = size(imread(auxv));
            else
                auxv = 'seleccion7.bmp';
            end
        else
            auxv = 'simbolo.bmp';
```

```
        end



    case 8
        if get(handles.booleanALG3,'Value')
            if opcion
                auxv = get(handles.archivoA3, 'String');
               tam = size(imread(auxv));
            else
                auxv = 'seleccion8.bmp';
            end
        else
            auxv = 'simbolo.bmp';
        end

end


% Representacion de la imagen seleccionada:
imshow(auxv);

if opcion

    % Visualizacion del tamaño de las imagenes (es imprescindible que sea el
    % mismo para todas):

    cad = strcat('Tamaño de la imagen:_',num2str(tam(1,1)));
    cad = strcat(cad,'_x_');
    cad = strcat(cad,num2str(tam(1,2)));

    % Se escribe en la linea de indicaciones:
    set(handles.texto,'String',cad);
end
```