Apéndice A

CÓDIGO MATLAB DE LAS FUNCIONES

seg.m

```
% *** ------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion principal de la implementacion realizada. Recibe a la entrada, entre otros                                | *** %
% ***| parametros, la imagen original a segmentar                                                                        | *** %
% *** ------------------------------------------------------------------------------------------------------------------------ *** %


function seg(filename,margen,flag_marco,color_option)

[color_camino,color_puntos] = colores(color_option);

tic;                                                    % Inicio t.segmentacion

[imag,map] = imread(filename);                          % Lectura de la imagen
[filas,columnas,dim] = size(imag);
image(imag);

if flag_marco == 1                % Si la opcion de marco de la imagen como camino minimo esta activada
   imag = marco_minimo(imag,filas,columnas);
end

[click_x,click_y,button] = ginput(1);                   % Captura pixel semilla
semilla = [round(click_y),round(click_x)];
cont_clicks = 1;                                        % Contador de puntos proporcionados por el usuario

imag(semilla(1),semilla(2),[1 2 3]) = color_puntos;
image(imag);

while button ~= 3                 % Se pediran puntos mientras el usuario no pinche el boton derecho

   [click_x,click_y,button] = ginput(1);                % Captura resto de pixeles
   posicion = [round(click_y),round(click_x)];

   if button ~= 3                 % Comprobacion de que no se pulso el boton derecho para finalizar la
                                   segmentacion

      % Formacion de la subimagen y su respectivo Mapa de Coste Local (MCL)
      [mapa_coste_local,sem_rel,pos_rel,vector_margenes] = mcl(imag,semilla,posicion,margen);

      % Creacion de la matriz 3D 'flechas_sub' que contiene el camino minimo de la subimagen
      flechas_sub = livewire(mapa_coste_local,sem_rel);

      % Adaptacion y Preparacion de la matriz 3D 'flechas' que contiene el camino minimo en el interior de
         la subimagen
      flechas =
        adaptacion_flechas(flechas_sub,semilla,posicion,sem_rel,pos_rel,filas,columnas,vector_margenes);

      % Llamada a la funcion que recorrera el camino minimo
      imag = camino(imag,flechas,semilla,posicion,color_camino,color_puntos);
```

```
        semilla = posicion;          % El posterior click pasa a ser ahora la nueva 'semilla'

        cont_clicks = cont_clicks + 1;

    end

end

timer = toc;                    % Fin t.segmentacion

imshow(imag);                % Presentacion final de la imagen segmentada

% Llamada a la funcion que crea los archivos resultado
crear_resultados(filename,imag,filas,columnas,margen,timer,cont_clicks);
```

---

colores.m

```
% *** ------------------------------------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion para la eleccion de colores que presentara la segmentacion de la imagen                                                                    | *** %
% *** ------------------------------------------------------------------------------------------------------------------------------------------------------ *** %

function [color_camino,color_puntos] = colores(color_option)

if color_option == 'b&y'

    color_camino = [0 0 255];
    color_puntos = [255 255 0];

elseif color_option == 'y&b'

    color_camino = [255 255 0];
    color_puntos = [0 0 255];

elseif color_option == 'r&g'

    color_camino = [255 0 0];
    color_puntos = [0 255 0];

elseif color_option == 'g&r'

    color_camino = [0 255 0];
    color_puntos = [255 0 0];

elseif color_option == 'b&w'

    color_camino = [0 0 0];
    color_puntos = [255 255 255];

elseif color_option == 'w&b'

    color_camino = [255 255 255];
    color_puntos = [0 0 0];

elseif color_option == 'w&g'

    color_camino = [255 255 255];
    color_puntos = [0 255 0];

end
```

marco_minimo.m

```
% *** ------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion que asigna valor nulo a los pixeles que forman el marco de la imagen                                        | *** %
% *** ------------------------------------------------------------------------------------------------------------------------ *** %

function  imagen = marco_minimo(imagen,filas,columnas)

imagen(1,1:columnas) = 0;              % marco arriba

imagen(filas,1:columnas) = 0;          % marco abajo

imagen(1:filas,1) = 0;                 % marco izquierda

imagen(1:filas,columnas) = 0;          % marco derecha
```

mcl.m

```
% *** ------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion que crea la subimagen correspondiente y devuelve su Mapa de Coste Local,                                    | *** %
% ***| asi como las posiciones relativas a la subimagen de los pixeles semilla y posicion                                  | *** %
% *** ------------------------------------------------------------------------------------------------------------------------ *** %

function [L,sem_rel,pos_rel,margenes] = mcl(imag,semilla,posicion,dist_margen)

wz = 7/10;                     % Pesos de las tres funciones de coste
wg = 2/10;
wd = 1/10;

im = double(imag);
im_R = double(imag(:,:,1));            % Lectura de las componentes RGB
im_G = double(imag(:,:,2));
im_B = double(imag(:,:,3));

% Creacion de la Sub-Imagen de la imagen y de cada componente RGB
[imag,sem_rel,pos_rel,margenes,ini_sub,fin_sub] = subimagen(im,semilla,posicion,dist_margen);
imag_R = im_R(ini_sub(1):fin_sub(1) , ini_sub(2):fin_sub(2));
imag_G = im_G(ini_sub(1):fin_sub(1) , ini_sub(2):fin_sub(2));
imag_B = im_B(ini_sub(1):fin_sub(1) , ini_sub(2):fin_sub(2));

%
_____________________________________________________________________________________
%                       Matriz de la funcion de 'Zero-crossing'

fz_R = edge(imag_R,'canny');           % Deteccion de bordes en las 3 componentes
fz_G = edge(imag_G,'canny');
fz_B = edge(imag_B,'canny');

fz_inv = fz_R | fz_G | fz_B;           % Con un '1' en una componente ya tenemos frontera
fz = 1 - double(fz_inv);               % Invirtiendo la salida de 'edge'

%
_____________________________________________________________________________________
%                     Matriz de la funcion Magnitud del Gradiente

fg_R = grad_mag(imag_R);               % Calculo de las 3 Matrices de la Magnitud del Gradiente
fg_G = grad_mag(imag_G);               % en las 3 componentes
fg_B = grad_mag(imag_B);

fg = max(max(fg_R,fg_G),fg_B);         % Solo se tendra en cuenta el valor minimo entre R,G,B
```

```
%
%_____________________________________________________________________
%                    Matriz de la funcion Direccion del Gradiente

fd = grad_dir(imag);

%
%_____________________________________________________________________
%                          MATRIZ DE COSTE LOCAL

[filas,columnas,dim] = size(imag);

for i = 1:filas                % Bucle para la formacion del Mapa de Coste Local (MCL)
   for j = 1:columnas

      L{i,j} = inf*ones(3,3);          % Inicializacion de las celdas de L (MCL)

      ini_fila = 1;                    % Indices necesarios en la creacion de la matriz del MCL
      fin_fila = 3;
      ini_columna = 1;
      fin_columna = 3;

      arriba = i - 1;
      abajo = i + 1;
      izquierda = j - 1;
      derecha = j + 1;

      if i == 1                        % Si nos encontramos en los bordes1ª y ultima fila
         ini_fila = 3;
         arriba = i + 1;
      elseif i == filas
         fin_fila = 1;
         abajo = i - 1;
      end

      if j == 1                        % Si nos encontramos en los bordes1ª y ultima columna
         ini_columna = 3;
         izquierda = j + 1;
      elseif j == columnas
         fin_columna = 1;
         derecha = j - 1;
      end

      %Formacion de los costes del pixel (i,j) hacia sus D-vecinos
      L{i,j}(ini_fila:2:fin_fila,ini_columna:2:fin_columna) = ...
               wd*fd{i,j}(ini_fila:2:fin_fila,ini_columna:2:fin_columna) +
               wz*fz(arriba:2:abajo,izquierda:2:derecha) +    wg*fg(arriba:2:abajo,izquierda:2:derecha);

      %Formacion de los costes del pixel (i,j) hacia sus 4-vecinos
      if i ~= 1
         L{i,j}(1,2) = wd*fd{i,j}(1,2) + wz*fz(i-1,j) + wg*fg(i-1,j);        % vecino arriba
      end
      if i ~= filas
         L{i,j}(3,2) = wd*fd{i,j}(3,2) + wz*fz(i+1,j) + wg*fg(i+1,j);        % vecino abajo
      end
      if j ~= 1
         L{i,j}(2,1) = wd*fd{i,j}(2,1) + wz*fz(i,j-1) + wg*fg(i,j-1);        % vecino izquierda
      end
      if j ~= columnas
         L{i,j}(2,3) = wd*fd{i,j}(2,3) + wz*fz(i,j+1) + wg*fg(i,j+1);        % vecino derecha
      end

   end
end
```

subimagen.m

```
% *** ------------------------------------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion que devuelve la subimagen rectangular doble de la contenida entre 'semilla'                                                                  | *** %
% ***| y 'posicion', y los indices relativos de semilla y posicion en dicha subimagen                                                                       | *** %
% *** ------------------------------------------------------------------------------------------------------------------------------------------------------ *** %

function [submapa,sem_rel,pos_rel,margenes,ini_sub,fin_sub] =
            subimagen(mapa,semilla,posicion,dist_margen)

dist_default = 10;                      % Distancia minima de la subimagen
dist_margen_y = dist_margen;
dist_margen_x = dist_margen;

% Comparando las coordenadas Y de 'semilla' y 'posicion' (indices de fila)
if semilla(1) <= posicion(1)

   ini_y = semilla(1);
   dist_y = posicion(1) - semilla(1);
   fin_y = ini_y + dist_y;
   if dist_y < dist_default             % Si 'dist_y' no llega al minimo exigido
      dist_margen_y = dist_margen + dist_default;
   end
   margen_y = ceil(dist_margen_y);

else

   ini_y = posicion(1);
   dist_y = semilla(1) - posicion(1);
   fin_y = ini_y + dist_y;
   if dist_y < dist_default             % Si 'dist_y' no llega al minimo exigido
      dist_margen_y = dist_margen + dist_default;
   end
   margen_y = ceil(dist_margen_y);

end

% Comparando las coordenadas X de 'semilla' y 'posicion' (indices de columna)
if semilla(2) <= posicion(2)

   ini_x = semilla(2);
   dist_x = posicion(2) - semilla(2);
   fin_x = ini_x + dist_x;
   if dist_x < dist_default             % Si 'dist_x' no llega al minimo exigido
      dist_margen_x = dist_margen + dist_default;
   end
   margen_x = ceil(dist_margen_x);

else

   ini_x = posicion(2);
   dist_x = semilla(2) - posicion(2);
   fin_x = ini_x + dist_x;
   if dist_x < dist_default             % Si 'dist_x' no llega al minimo exigido
      dist_margen_x = dist_margen + dist_default;
   end
   margen_x = ceil(dist_margen_x);

end

ini = [ini_y,ini_x];                    % Formacion de vectores de interes para la funcion 'comprueba'
fin = [fin_y,fin_x];
margen = [margen_y,margen_x];
dimensiones = [size(mapa,1),size(mapa,2)];
```

```
dist = [dist_y,dist_x];

% Llamada a la funcion que delimitara la subimagen aumentada y comprobara que no salga de las
% dimensiones de la imagen original

[ini_sub,fin_sub,sem_rel,pos_rel,margenes] =
        comprueba(dimensiones,ini,fin,margen,semilla,posicion,dist);

submapa = mapa(ini_sub(1):fin_sub(1) , ini_sub(2):fin_sub(2));          % Ampliacion de la subimagen
```

---

comprueba.m

```
% *** ------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion que comprueba que el tamaño de la subimagen no excede el de la imagen              | *** %
% ***| original y, en caso negativo, delimitira la misma a las dimensiones originales            | *** %
% *** ------------------------------------------------------------------------------------------------------------------------ *** %

function [ini_sub,fin_sub,sem_rel,pos_rel,margenes] =
        comprueba(dimensiones,ini,fin,margen,semilla,posicion,dist)

filas = dimensiones(1);
columnas = dimensiones(2);

if ini(1) - margen(1) < 1                   % sobrepasa por arriba
   ini_sub(1) = 1;
   if semilla(1) <= posicion(1)
      sem_rel(1) = semilla(1);
      pos_rel(1) = semilla(1) + dist(1);
      margen_arr = semilla(1) - 1;
   else
      pos_rel(1) = posicion(1);
      sem_rel(1) = posicion(1) + dist(1);
      margen_arr = posicion(1) - 1;
   end
else                                        % NO sobrepasa por arriba
   ini_sub(1) = ini(1) - margen(1);
   margen_arr = margen(1);
   if semilla(1) <= posicion(1)
      sem_rel(1) = 1 + margen(1);
      pos_rel(1) = sem_rel(1) + dist(1);
   else
      pos_rel(1) = 1 + margen(1);
      sem_rel(1) = pos_rel(1) + dist(1);
   end
end

if fin(1) + margen(1) > filas               % sobrepasa por abajo
   fin_sub(1) = filas;
   if semilla(1) <= posicion(1)
      margen_aba = filas - posicion(1);
   else
      margen_aba = filas - semilla(1);
   end
else                                        % NO sobrepasa por abajo
   fin_sub(1) = fin(1) + margen(1);
   margen_aba = margen(1);
end

if ini(2) - margen(2) < 1                   % sobrepasa por la izquierda
   ini_sub(2) = 1;
   if semilla(2) <= posicion(2)
      sem_rel(2) = semilla(2);
```

```
        pos_rel(2) = semilla(2) + dist(2);
        margen_izq = semilla(2) - 1;
    else
        pos_rel(2) = posicion(2);
        sem_rel(2) = posicion(2) + dist(2);
        margen_izq = posicion(2) - 1;
    end
else                                    % NO sobrepasa por la izquierda
    ini_sub(2) = ini(2) - margen(2);
    margen_izq = margen(2);
    if semilla(2) <= posicion(2)
        sem_rel(2) = 1 + margen(2);
        pos_rel(2) = sem_rel(2) + dist(2);
    else
        pos_rel(2) = 1 + margen(2);
        sem_rel(2) = pos_rel(2) + dist(2);
    end
end

if fin(2) + margen(2) > columnas        % sobrepasa por la derecha
    fin_sub(2) = columnas;
    if semilla(2) <= posicion(2)
        margen_der = columnas - posicion(2);
    else
        margen_der = columnas - semilla(2);
    end
else                                    % NO sobrepasa por la derecha
    fin_sub(2) = fin(2) + margen(2);
    margen_der = margen(2);
end

% Vector con los distintos margenes de la subimagen
margenes = [margen_arr,margen_aba,margen_izq,margen_der];
```

---

grad_mag.m

```
% *** ------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion que devuelve la Matriz de la funcion de la Magnitud del Gradiente                                          | *** %
% *** ------------------------------------------------------------------------------------------------------------------------ *** %

function matriz_fg = grad_mag(imag)


[Gx,Gy] = gradient(imag);               % Funcion que calcula el gradiente de una matriz

G = sqrt(Gx.^2 + Gy.^2) + eps;


% Se asociaran costes bajos a magnitudes de gradiente grandes, ya que estas indican cambios
% muy significativos de intensidad, lo que se traduce en la presencia de fronteras

matriz_fg = 1- G/max(max(G));           % fg = [0 , ~1]
```

grad_dir.m

```
% *** ------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion que devuelve la Matriz de la funcion de la Direccion del gradiente                                          | *** %
% *** ------------------------------------------------------------------------------------------------------------------------ *** %

function matriz_fd = grad_dir(imag)

[filas,columnas] = size(imag);

[Gx,Gy] = gradient(imag);

G = sqrt(Gx.^2 + Gy.^2) + eps;
Gx_n = Gx./G;                          % Gradientes normalizados
Gy_n = Gy./G;

for i = 1:filas
          for j = 1:columnas

    Dp = [Gy_n(i,j),-Gx_n(i,j)];                % Vector direccion del pixel p(i,j), perpendicular al gradiente

    for dy = -1:1
       if (i+dy) >= 1 & (i+dy) <= filas
          for dx = -1:1
             if (j+dx) >= 1 & (j+dx) <= columnas
                if dy ~= 0 | dx ~= 0                    % 8-vecinos del pixel en cuestion

                   Dq = [Gy_n(i+dy,j+dx),-Gx_n(i+dy,j+dx)];    % Vector direccion del pixel q(i+dx,j+dy),
                                                                %   perpendicular al gradiente
                   dr = [dy,dx];                            % Vector incremento

                   Lpq = dot(Dp,dr);                        % Vector direccion entre p y q
                   if Lpq >= 0
                      Lpq = dr;
                   else
                      Lpq = -dr;
                   end
                   Lpq_n = Lpq/(sqrt(Lpq(1)^2 + Lpq(2)^2));    % Vector direccion entre p y q normalizado

                   dp = dot(Dp,Lpq_n);
                   dq = dot(Lpq_n,Dq);

                   fd = (1/pi)*(acos(dp) + acos(dq));          % Valor de la direccion del gradiente entre p y q
                   bloque(dy+2,dx+2) = fd;

                end
             end
          end
       end
    end

    matriz_fd{i,j} = bloque;

  end
end
```

livewire.m

```
% *** ---------------------------------------------------------------------------------------------------------------------- *** %
% ***| Funcion que crea el Mapa de Coste Acumulado (MCA) a partir del Mapa  Coste Local (MCL) | *** %
% ***| y define la matriz 'flechas' que contiene el camino minimo desde un punto al resto    | *** %
% *** ---------------------------------------------------------------------------------------------------------------------- *** %

function flechas = livewire(mapa_coste_local,semilla)

[filas,columnas] = size(mapa_coste_local);

flechas = zeros(filas,columnas,2);                  % Posiciones de los pixeles que recorren el camino
                                                      minimo
expanded = zeros(filas,columnas);                   % Matriz donde se marcan a '1' los pixeles ya
                                                      Expandidos
active = zeros(filas,columnas);                     % Matriz donde se marcan a '1' los pixeles que forman
                                                      parte de la Lista Activa
mapa_coste_acumulado = inf*ones(filas,columnas);          % Mapa de Coste Acumulado desde el pixel
                                                            'semilla' al resto

mapa_coste_acumulado(semilla(1),semilla(2)) = 0;
expanded(semilla(1),semilla(2)) = 1;

vecinos = ver_vecinos(semilla,filas,columnas);          % Vector que define los 8-vecinos del pixel 'semilla'

for v = vecinos

   active(semilla(1)+v(1),semilla(2)+v(2)) = 1 ;          % Se marcan los pixeles candidatos a expandirse
                                                            (8-vecinos del expandido)

   mapa_coste_acumulado(semilla(1)+v(1),semilla(2)+v(2)) = ...
      sqrt(abs(v(1))+abs(v(2)))*mapa_coste_local{semilla(1),semilla(2)}(v(1)+2,v(2)+2);

   flechas(semilla(1)+v(1),semilla(2)+v(2),1) = semilla(1);    % Se coge la posicion del pixel que sigue al
                                                                 actual
   flechas(semilla(1)+v(1),semilla(2)+v(2),2) = semilla(2);

end

% Calculando el Mapa de Coste Acumulado para todos los pixeles no INF
while sum(sum(expanded)) < filas*columnas

   q = minimo(mapa_coste_acumulado,active);                % Busca el siguiente pixel a expandir
                                                           % (minimo de la Lista Activa)
   vecinos = ver_vecinos(q,filas,columnas);                % Vector que define los 8-vecinos del pixel q

   expanded(q(1),q(2)) = 1;                                % Se marca como pixel Expandido
   active(q(1),q(2)) = 0;                                  % Se quita de la Lista Activa

   for v = vecinos

      if expanded(q(1)+v(1),q(2)+v(2)) == 0                % pixeles no Expandidos aun

         % MCA(q) = min{MCA(q)antiguo , MCL(p,q)escalado + MCA(p)}    (Actualizando el MCA)
         if mapa_coste_acumulado(q(1)+v(1),q(2)+v(2)) > ...
               sqrt(abs(v(1))+abs(v(2)))*mapa_coste_local{q(1),q(2)}(v(1)+2,v(2)+2) +
               mapa_coste_acumulado(q(1),q(2))

            mapa_coste_acumulado(q(1)+v(1),q(2)+v(2)) = ...
               sqrt(abs(v(1))+abs(v(2)))*mapa_coste_local{q(1),q(2)}(v(1)+2,v(2)+2) +
               mapa_coste_acumulado(q(1),q(2));

            active(q(1)+v(1),q(2)+v(2)) = 1;              % Se marcan los pixeles candidatos a expandirse
                                                          % (8-vecinos del expandido no expandidos aun)
```

```
            flechas(q(1)+v(1),q(2)+v(2),1) = q(1);        % Se coge la posicion del pixel que sigue al actual
            flechas(q(1)+v(1),q(2)+v(2),2) = q(2);

         end
      end

   end
end
```

---

ver_vecinoss.m

```
% *** ---------------------------------------------------------------------------------------------------------------------------------- *** %
% ***| Funcion que determina los vecinos de un pixel de la imagen                                                                    | *** %
% *** ---------------------------------------------------------------------------------------------------------------------------------- *** %

function vecinos = ver_vecinos(q,filas,columnas)


if q(1) == 1
   vecinos = [ 0 1 1 1 0 ; -1 -1 0 1 1 ];

elseif q(1) == filas
   vecinos = [ 0 -1 -1 -1 0 ; -1 -1 0 1 1 ];

elseif q(2) == 1
   vecinos = [ -1 -1 0 1 1 ; 0 1 1 1 0 ];

elseif q(2) == columnas
   vecinos = [ 1 1 0 -1 -1 ; 0 -1 -1 -1 0 ];

else
   vecinos = [ 1 1 0 -1 -1 -1 0 1 ; 0 1 1 1 0 -1 -1 -1 ];

end


if q(1) == 1 & q(2) == 1
   vecinos = [ 0 1 1 ; 1 1 0 ];

elseif q(1) == 1 & q(2) == columnas
   vecinos = [ 0 1 1 ; -1 -1 0 ];

elseif q(1) == filas & q(2) == 1
   vecinos = [ -1 -1 0 ; 0 1 1 ];

elseif q(1) == filas & q(2) == columnas
   vecinos = [ 0 -1 -1 ; -1 -1 0 ];

end
```

minimo.m

```
% *** ------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion que devuelve el siguiente pixel a expandir (minimo de la Lista Activa)                                    | *** %
% *** ------------------------------------------------------------------------------------------------------------------------ *** %

function q = minimo(mapa_coste_acumulado,lista_activa)

[x,y] = find(lista_activa);                    % Indices de los elementos no nulos de la Lista Activa

pixel_minimo_actual = inf;

% Bucle que va buscando el pixel con valor menor de los pertenecientes a la Lista Activa
for k = 1:length(x)

   if mapa_coste_acumulado(x(k),y(k)) < pixel_minimo_actual

      pixel_minimo_actual = mapa_coste_acumulado(x(k),y(k));
      q = [x(k),y(k)];

   end

end
```

adaptacion_flechas.m

```
% *** ------------------------------------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion que copia la matriz flechas de la subimagen ('flechas_sub') en la de la                                                                    | *** %
% ***| imagen completa ('flechas')                                                                                                                        | *** %
% ***|                                                                                                                                                    | *** %
% ***| 'margenes'[1 2 3 4] = [margen arriba, margen abajo, margen izquierda, margen derecha]                                                               | *** %
% *** ------------------------------------------------------------------------------------------------------------------------------------------------------ *** %

function flechas =
        adaptacion_flechas(flechas_sub,semilla,posicion,sem_rel,pos_rel,filas,columnas,margenes)

flechas = inf*ones(filas,columnas,2);

if semilla(1) <= posicion(1) & semilla(2) <= posicion(2)          % 'semilla' en esquina superior izquierda

   ini_fila = semilla(1) - margenes(1);
   fin_fila = posicion(1) + margenes(2);
   ini_columna = semilla(2) - margenes(3);
   fin_columna = posicion(2) + margenes(4);

   flechas(ini_fila:fin_fila,ini_columna:fin_columna,1) = flechas_sub(:,:,1) + semilla(1) - sem_rel(1);
   flechas(ini_fila:fin_fila,ini_columna:fin_columna,2) = flechas_sub(:,:,2) + semilla(2) - sem_rel(2);

elseif semilla(1) <= posicion(1) & posicion(2) <= semilla(2)      % 'semilla' en esquina superior derecha

   ini_fila = semilla(1) - margenes(1);
   fin_fila = posicion(1) + margenes(2);
   ini_columna = posicion(2) - margenes(3);
   fin_columna = semilla(2) + margenes(4);

   flechas(ini_fila:fin_fila,ini_columna:fin_columna,1) = flechas_sub(:,:,1) + semilla(1) - sem_rel(1);
   flechas(ini_fila:fin_fila,ini_columna:fin_columna,2) = flechas_sub(:,:,2) + posicion(2) - pos_rel(2);

elseif posicion(1) <= semilla(1) & posicion(2) <= semilla(2)      % 'semilla' en esquina inferior derecha

   ini_fila = posicion(1) - margenes(1);
   fin_fila = semilla(1) + margenes(2);
   ini_columna = posicion(2) - margenes(3);
   fin_columna = semilla(2) + margenes(4);

   flechas(ini_fila:fin_fila,ini_columna:fin_columna,1) = flechas_sub(:,:,1) + posicion(1) - pos_rel(1);
   flechas(ini_fila:fin_fila,ini_columna:fin_columna,2) = flechas_sub(:,:,2) + posicion(2) - pos_rel(2);

elseif posicion(1) <= semilla(1) & semilla(2) <= posicion(2)      % 'semilla' en esquina inferior izquierda

   ini_fila = posicion(1) - margenes(1);
   fin_fila = semilla(1) + margenes(2);
   ini_columna = semilla(2) - margenes(3);
   fin_columna = posicion(2) + margenes(4);

   flechas(ini_fila:fin_fila,ini_columna:fin_columna,1) = flechas_sub(:,:,1) + posicion(1) - pos_rel(1);
   flechas(ini_fila:fin_fila,ini_columna:fin_columna,2) = flechas_sub(:,:,2) + semilla(2) - sem_rel(2);

end
```

camino.m

```
% *** ------------------------------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion que recorre el camino minimo entre semilla y posicion                                                                                | *** %
% *** ------------------------------------------------------------------------------------------------------------------------------------------------ *** %

function imag = camino(imag,flechas,semilla,posicion,color_camino,color_puntos)

posicion_original = posicion;

image(imag);

% Recorre el camino hacia atras, desde el ultimo click hasta la semilla
while (posicion(1) ~= semilla(1) | posicion(2) ~= semilla(2))

    posicion_nueva(1) = flechas(posicion(1),posicion(2),1);
    posicion_nueva(2) = flechas(posicion(1),posicion(2),2);

    posicion(1) = posicion_nueva(1);
    posicion(2) = posicion_nueva(2);

    imag(posicion(1),posicion(2),[1 2 3]) = color_camino;          % marcado de cada pixel del camino

end

imag(semilla(1),semilla(2),[1 2 3]) = color_puntos;           % marcado de los 2 puntos que se han pinchado
imag(posicion_original(1),posicion_original(2),[1 2 3]) = color_puntos;
image(imag);
```

crear_resultados.m

```
% *** ------------------------------------------------------------------------------------------------------------------------------------------------ *** %
% ***| Funcion que crea los archivos de salida 'results.txt' y 'results.bmp' con datos de                                                           | *** %
% ***| interes del proceso de segmentacion y la imagen segmentada respectivamente                                                                   | *** %
% *** ------------------------------------------------------------------------------------------------------------------------------------------------ *** %

function crear_resultados(filename,imag,filas,columnas,dist_default,timer,cont_clicks)

date = clock;                          % Captura del reloj

fid = fopen('results.txt','wt+');      % Apertura del fichero de escritura

% Escribiendo datos en el archivo de texto
fprintf(fid,'\n %d - %d - %d                          %d:%d:%2.0f',date(3),date(2),date(1),date(4),date(5),date(6));
fprintf(fid,'\n_____________________________________________________________________________ \n\n');

fprintf(fid,'\n ****************************          DATA          **************************** \n\n');
fprintf(fid,'\n Original image:   %s \n\n Size (px):   %d x %d \n',filename,columnas,filas);
fprintf(fid,'\n Marginal distance:   %d \n',dist_default);


fprintf(fid,'\n\n ****************************          RESULTS          ********************************** \n' );
fprintf(fid,'\n Number of points:   %d \n',cont_clicks);
fprintf(fid,'\n Segmentation time:   %4.4f seconds \n',timer);

fclose(fid);                           % Cierre del fichero de escritura


imwrite(imag,'results.bmp');           % Creacion del archivo imagen
```