Apéndice E. Código

Se exponen los códigos utilizados en esta tesis. En una primera parte se explican todas las funciones utilizadas, aquellas que son parte de MATLAB no se comentan en este apéndice, estas pueden ser consultadas en la documentación de ayuda de MATLAB. Seguido de esto se presentan los programas que en general son llamadas a funciones con algunas operaciones entre estas llamadas que son comentadas pertinentemente.

E.1 Funciones generales

Se exponen a continuación las funciones utilizadas para los diferentes modelos y el caso de experimentación.

E.1.1 Función gaussianBlur

Esta función suaviza la imagen acorde a lo mostrado en el capítulo 2.

```
function GI = gaussianBlur(I,s)
%     GAUSSIANBLUR suaviza la imagen
%     GI = gaussianBlur(I,s)
%     I es la imagen, s es la desviacion estandar
%     GI es la imagen desvanecida

%    Chenyang Xu and Jerry L. Prince 6/17/97
%    Copyright (c) 1996-97 by Chenyang Xu and Jerry L. Prince

M = gaussianMask(1,s);
M = M/sum(sum(M));   % normalizar la mascara para que su valor sea igual a 1

GI = xconv2(I,M); %convolucion de la imagen original con la mascara M.
```

E.1.1.1 Función gaussianMask

```
function M = gaussianMask(k,s)
% k: el factor de escala
% s: desviacion estandar

R = ceil(3*s); % radio truncado
for i = -R:R,
   for j = -R:R,
      M(i+ R+1,j+R+1) = k * exp(-(i*i+j*j)/2/s/s)/(2*pi*s*s);
   end
end
```

E.1.2 Función manual

Genera la interfase para la ubicación de los puntos de control.

```
function [x,y]=manual(imagen);

% MANUAL realiza la operacion manual de ubicar
% los puntos de control iniciales alrededor del
% contorno que deseamos cerrar.

% imagen es correspondiente a la imagen original
% con la que vamos a trabajar.
% x, y son los puntos en la imagen correspondientes
% a los puntos de control.

% Miguel Angel Torres Tello 20/01/06

x=[];   %se inicializan variables
y=[];
x2=0;
y2=0;
i=0;
boton=1;

figure, imshow(imagen);
title('De click al boton izq. del mouse para puntos de control. Para finalizar click con boton der.');
hold on;

while boton==1

   [x2,y2,boton]=ginput(1);    %obtencion de puntos
   x=[x;x2];
   y=[y;y2];
   plot(x,y,'rs-')
   x2=x;
   y2=y;
   i=i+1;

end;

plot([x2(i),x(1)],[y2(i),y(1)],'rs-')
hold off;
```

E.1.3 Función snakedisp2

Para mostrar el estado actual de la curva

```
function snakedisp2(x,y,style)
%muestra el resultado actual de la curva

hold on
x = x(:); y = y(:);
plot([x;x(1,1)],[y;y(1,1)],style,'LineWidth',3);
hold off
```

E.1.4 Función snakedeform

Esta función deforma el modelo activo acorde a la definición de Kass

```
function [x,y] = snakedeform(x,y,alpha,beta,gamma,kappa,fx,fy,ITER)
% SNAKEDEFORM  Deforma el modelo activo acorde a la fuerzas externas
%     [x,y] = snakedeform(x,y,alpha,beta,gamma,kappa,fx,fy,ITER)
%
%     alpha:   parametro de tension
%     beta:    parametro de rigidez
%     gamma:   parametro de control del punto de control anterior
%     kappa:   parametro de fuerza externa
%     fx,fy:   campo de fuerza externa

%      Chenyang Xu and Jerry L. Prince, 4/1/95, 6/17/97
%      Copyright (c) 1995-97 by Chenyang Xu and Jerry L. Prince
%      Image Analysis and Communications Lab, Johns Hopkins University


% generar los parametros adecuados para la curva

N = length(x);

alpha = alpha* ones(1,N);
beta = beta*ones(1,N);

% producir cinco vectores diagonales
alpham1 = [alpha(2:N) alpha(1)];
alphap1 = [alpha(N) alpha(1:N-1)];
betam1 = [beta(2:N) beta(1)];
betap1 = [beta(N) beta(1:N-1)];

a = betam1;
b = -alpha - 2*beta - 2*betam1;
c = alpha + alphap1 +betam1 + 4*beta + betap1;
d = -alphap1 - 2*beta - 2*betap1;
e = betap1;

% generar la matriz pentadiagonal
A = diag(a(1:N-2),-2)+ diag(a(N-1:N),N-2);
A = A + diag(b(1:N-1),-1) + diag(b(N), N-1);
A = A + diag(c);
A = A + diag(d(1:N-1),1) + diag(d(N),-(N-1));
A = A + diag(e(1:N-2),2) + diag(e(N-1:N),-(N-2));

invAI = inv(A + gamma * diag(ones(1,N)));

for count = 1:ITER,
  vfx = interp2(fx,x,y,'*linear');
  vfy = interp2(fy,x,y,'*linear');

  % deformar el modelo activo
  x = invAI * (gamma* x + kappa*vfx);
  y = invAI * (gamma* y + kappa*vfy);
end
```

E.1.5 Función snakeinterp

Se realiza la interpolación de la curva utilizando umbrales para coordinar distancias

```
function [xi,yi] = snakeinterp(x,y,dmax,dmin)
%SNAKEINTERP  realiza la interpolacion adecuada
%   [xi,yi] = snakeinterp(x,y,dmax,dmin)
```

```
%
%   dmax: maxima distancia entre dos puntos de control
%   dmin: minimi distancia entre dos puntos de control
%   d(i,i+1)>dmax, se agrega un nuevo punto
%   d(i,i+1)<dmin, tanto i como i+1 se remueven
%

%   Chenyang Xu and Jerry L. Prince, 4/1/95, 6/17/97
%   Copyright (c) 1995-97 by Chenyang Xu and Jerry L. Prince
%   Image Analysis and Communications Lab, Johns Hopkins University

% convertir a vector columna
x = x(:); y = y(:);

N = length(x);

d = abs(x([2:N 1])- x(:)) + abs(y([2:N 1])- y(:));

% remover los puntos que tengan distancias menores a dmin
IDX = (d<dmin);

idx = find(IDX==0);
x = x(idx);
y = y(idx);

N = length(x);
d = abs(x([2:N 1])- x(:)) + abs(y([2:N 1])- y(:));

IDX = (d>dmax);

z = snakeindex(IDX);

p = 1:N+1;

xi = interp1(p,[x;x(1)],z');
yi = interp1(p,[y;y(1)],z');

N = length(xi);
d = abs(xi([2:N 1])- xi(:)) + abs(yi([2:N 1])- yi(:));

while (max(d)>dmax),

   IDX = (d>dmax);
   z = snakeindex(IDX);

   p = 1:N+1;

   xi = interp1(p,[xi;xi(1)],z');
   yi = interp1(p,[yi;yi(1)],z');

   N = length(xi);
   d = abs(xi([2:N 1])- xi(:)) + abs(yi([2:N 1])- yi(:));
end
```

E.1.6 Función snakedeform2

Esta función deforma el modelo activo acorde a la definición de L. Cohen.

```
function [x,y] = snakedeform(x,y,alpha,beta,gamma,kappa, kappap, fx,fy,ITER)
% SNAKEDEFORM2  Deforma el modelo activo acorde a la fuerzas externas y
```

```
% anexando la fuerza del modelo de L.Cohen

%     [x,y] = snakedeform(x,y,alpha,beta,gamma,kappa,fx,fy,ITER)
%
%     alpha:   parametro de tension
%     beta:    parametro de rigidez
%     gamma:   parametro de control del punto de control anterior
%     kappa:   parametro de fuerza externa
%     kappap:  parametro de fuerza de presion
%     fx,fy:   campo de fuerza externa

%      Chenyang Xu and Jerry L. Prince, 4/1/95, 6/17/97
%      Copyright (c) 1995-97 by Chenyang Xu and Jerry L. Prince
%      Image Analysis and Communications Lab, Johns Hopkins University


% generar los parametros adecuados para la curva

N = length(x);

alpha = alpha* ones(1,N);
beta = beta*ones(1,N);

% producir cinco vectores diagonales
alpham1 = [alpha(2:N) alpha(1)];
alphap1 = [alpha(N) alpha(1:N-1)];
betam1 = [beta(2:N) beta(1)];
betap1 = [beta(N) beta(1:N-1)];

a = betam1;
b = -alpha - 2*beta - 2*betam1;
c = alpha + alphap1 +betam1 + 4*beta + betap1;
d = -alphap1 - 2*beta - 2*betap1;
e = betap1;

% generar la matriz pentadiagonal
A = diag(a(1:N-2),-2) + diag(a(N-1:N),N-2);
A = A + diag(b(1:N-1),-1) + diag(b(N), N-1);
A = A + diag(c);
A = A + diag(d(1:N-1),1) + diag(d(N),-(N-1));
A = A + diag(e(1:N-2),2) + diag(e(N-1:N),-(N-2));

invAI = inv(A + gamma * diag(ones(1,N)));

for count = 1:ITER,
  vfx = interp2(fx,x,y,'*linear');
  vfy = interp2(fy,x,y,'*linear');

  % fuerza de presion
  xp = [x(2:N);x(1)];    yp = [y(2:N);y(1)];
  xm = [x(N);x(1:N-1)];  ym = [y(N);y(1:N-1)];

  qx = xp-xm;  qy = yp-ym;
  pmag = sqrt(qx.*qx+qy.*qy);
  px = qy./pmag;     py = -qx./pmag;

  % deformar el modelo activo
  x = invAI * (gamma* x + kappa*vfx + kappap.*px);
  y = invAI * (gamma* y + kappa*vfy + kappap.*py);
end
```

E.1.7 Función GVF.

Esta función calcula el campo GVF

```
function [u,v] = GVF(f, mu, ITER)
%GVF obtiene el campo GVF
%   [u,v] = GVF(f, mu, ITER)
%   f: mapa de borde
%   mu: coeficiente de regularizacion GVF
%   ITER: numero de iteraciones realizadas para el campo

%   Chenyang Xu and Jerry L. Prince 6/17/97
%   Copyright (c) 1996-99 by Chenyang Xu and Jerry L. Prince
%   Image Analysis and Communications Lab, Johns Hopkins University

[m,n] = size(f);
fmin  = min(f(:));
fmax  = max(f(:));
f = (f-fmin)/(fmax-fmin);  % Normalizar f en el rango de [0 1]

f = BoundMirrorExpand(f);  % Considerar condiciones de frontera


f=GaussianBlur(f,1);

[fx,fy] = gradient(f);     % Calcular el mapa de bordes
u = fx; v = fy;            % Inicializar GVF con el gradiente
SqrMagf = fx.*fx + fy.*fy; % Magnitud del campo gradiente

% Resolver iterativamente el campo GVF, u y v
for i=1:ITER,
  u = BoundMirrorEnsure(u);
  v = BoundMirrorEnsure(v);
  u = u + mu*4*del2(u) - SqrMagf.*(u-fx);
  v = v + mu*4*del2(v) - SqrMagf.*(v-fy);
end

u = BoundMirrorShrink(u);
v = BoundMirrorShrink(v);
```

E.1.8 Código GVF2

Esta función calcula el campo GVF para imágenes con escala de grises.

```
function [u,v] = GVF2(f, mu, ITER)
%GVF obtiene el campo GVF para imagen con escala de grises
%   [u,v] = GVF(f, mu, ITER)
%   f: mapa de borde
%   mu: coeficiente de regularizacion GVF
%   ITER: numero de iteraciones realizadas para el campo

%   Chenyang Xu and Jerry L. Prince 6/17/97
%   Copyright (c) 1996-99 by Chenyang Xu and Jerry L. Prince
%   Image Analysis and Communications Lab, Johns Hopkins University
%   Miguel Angel Torres Tello 20/01/06 (Algoritmo para escala de grises)

[m,n] = size(f);
fmin  = min(f(:));
```

```
fmax  = max(f(:));
f = (f-fmin)/(fmax-fmin);  % Normalizar el rango de [0 1]

f = BoundMirrorExpand(f);  % Cuidar condiciones de frontera

% Calculo de fuerzas para imagenes con escala de grises
I2=gaussianBlur(f,1);
[Ix,Iy]=gradient(I2);
fx=gradient(abs(Ix).^(1/2));
Iy=permute(Iy,[2 1]);
py=gradient(abs(Iy).^(1/2));
fy=ipermute(py,[2 1]);

%**********************

u = fx; v = fy;            % Inicializar GVF
SqrMagf = fx.*fx + fy.*fy; % Magnitud de el campo GVF inicializado

% Resolver iterativamente el campo GVF, u y v
for i=1:ITER,
  u = BoundMirrorEnsure(u);
  v = BoundMirrorEnsure(v);
  u = u + mu*4*del2(u) - SqrMagf.*(u-fx);
  v = v + mu*4*del2(v) - SqrMagf.*(v-fy);
end

u = BoundMirrorShrink(u);
v = BoundMirrorShrink(v);
```

E.1.9 Funcion energy_term

Función para obtener la energía de término que se define en el capítulo 2.

```
% Copyright (c) 2002-2005 Peter Kovesi
% School of Computer Science & Software Engineering
% The University of Western Australia
% http://www.csse.uwa.edu.au/
%
% Permission is hereby granted, free of charge, to any person obtaining a copy
% of this software and associated documentation files (the "Software"), to deal
% in the Software without restriction, subject to the following conditions:
%
% The above copyright notice and this permission notice shall be included in
% all copies or substantial portions of the Software.
%
% The Software is provided "as is", without warranty of any kind.

% March    2002 - original version
% December 2002 - updated comments
% August   2005 - changed so that code calls nonmaxsuppts
function[px3,py3]= energy_term(I)

   dx = [-1 0 1; -1 0 1; -1 0 1];   % Derivative masks
   dy = dx';
   sigma=1;

   Ix = conv2(I, dx, 'same');      % Image derivatives
   Iy = conv2(I, dy, 'same');

   % Generate Gaussian filter of size 6*sigma (+/- 3sigma) and of
   % minimum size 1x1.
```

```
   g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);

   Ix2 = conv2(Ix.^2, g, 'same'); % Smoothed squared image derivatives
   Iy2 = conv2(Iy.^2, g, 'same');
   Ixy = conv2(Ix.*Iy, g, 'same');

   % Compute the Harris corner measure. Note that there are two measures
   % that can be calculated.  I prefer the first one below as given by
   % Nobel in her thesis (reference above).  The second one (commented out)
   % requires setting a parameter, it is commonly suggested that k=0.04 - I
   % find this a bit arbitrary and unsatisfactory.

   cim = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps); % My preferred  measure.
%    k = 0.04;
%    cim = (Ix2.*Iy2 - Ixy.^2) - k*(Ix2 + Iy2).^2; % Original Harris measure.

   [px3,py3]=gradient(cim);
```

E.1.10 Función snakedeform_ex

Función que se utiliza para deformar el modelo de Kass y Xu para experimentación generando un vector de error para definir un umbral.

```
function [x,y,x2,y2] = snakedeform_ex(x,y,alpha,beta,gamma,kappa,fx,fy,ITER)
% SNAKEDEFORM_EX  Deforma el modelo activo acorde a la fuerzas externas
%
%    [x,y,x2,y2] = snakedeform_ex(x,y,alpha,beta,gamma,kappa,fx,fy,ITER)
%
%    alpha:   parametro de tension
%    beta:    parametro de rigidez
%    gamma:   parametro de control del punto de control anterior
%    kappa:   parametro de fuerza externa
%    fx,fy:   campo de fuerza externa
%    x,y:     salida que representa la curva
%    x2,y2:   vectores de error correspondientes al avance del punto
%             anterior al actual
%     Chenyang Xu and Jerry L. Prince, 4/1/95, 6/17/97
%     Copyright (c) 1995-97 by Chenyang Xu and Jerry L. Prince
%     Image Analysis and Communications Lab, Johns Hopkins University
%   Miguel Angel Torres Tello 20/01/06 (Algoritmo para el vector de error)


% generar los parametros adecuados para la curva

N = length(x);

alpha = alpha* ones(1,N);
beta = beta*ones(1,N);

% produce los vectores diagonales
alpham1 = [alpha(2:N) alpha(1)];
alphap1 = [alpha(N) alpha(1:N-1)];
betam1 = [beta(2:N) beta(1)];
betap1 = [beta(N) beta(1:N-1)];

a = betam1;
b = -alpha - 2*beta - 2*betam1;
c = alpha + alphap1 +betam1 + 4*beta + betap1;
d = -alphap1 - 2*beta - 2*betap1;
e = betap1;
```

```
% genera la matriz pentadiagonal
A = diag(a(1:N-2),-2)+ diag(a(N-1:N),N-2);
A = A + diag(b(1:N-1),-1) + diag(b(N), N-1);
A = A + diag(c);
A = A + diag(d(1:N-1),1) + diag(d(N),-(N-1));
A = A + diag(e(1:N-2),2) + diag(e(N-1:N),-(N-2));

invAI = inv(A + gamma * diag(ones(1,N)));

for count = 1:ITER,
  vfx = interp2(fx,x,y,'*linear');
  vfy = interp2(fy,x,y,'*linear');

  temp_x=x;
  temp_y=y;

  % deforma el modelo activo
  x = invAI * (gamma* x + kappa*vfx);
  y = invAI * (gamma* y + kappa*vfy);

  x2=abs(temp_x-x);
  y2=abs(temp_y-y);
  x2=x2(:);
  y2=y2(:);
end
```

E.1.11 Función snakedeform_ex2

Función que se utiliza para deformar el modelo de Cohen para experimentación generando un vector de error para definir un umbral. En esta función se hace uso de la energía de término para localizar esquinas

```
function [x,y,x2,y2] = snakedeform_ex2(x,y,alpha,beta,gamma,kappa, kappap, fx,fy,ITER,px3,py3)
% SNAKEDEFORM_EX2  Deforma el modelo activo acorde a la fuerzas externas
%
%    [x,y,x2,y2] = snakedeform_ex(x,y,alpha,beta,gamma,kappa,fx,fy,ITER)
%
%    alpha:   parametro de tension
%    beta:    parametro de rigidez
%    gamma:   parametro de control del punto de control anterior
%    kappa:   parametro de fuerza externa
%    kappap:  parametro de fuerza de presion
%    fx,fy:   campo de fuerza externa
%    x,y:     salida que representa la curva
%    x2,y2:   vectores de error correspondientes al avance del punto
%             anterior al actual
%     Chenyang Xu and Jerry L. Prince, 4/1/95, 6/17/97
%     Copyright (c) 1995-97 by Chenyang Xu and Jerry L. Prince
%     Image Analysis and Communications Lab, Johns Hopkins University
%   Miguel Angel Torres Tello 20/01/06 (Algoritmo para el vector de error)


% generar los parametros adecuados para la curva

N = length(x);

alpha = alpha* ones(1,N);
beta = beta*ones(1,N);
```

```
% produce los vectores diagonales
alpham1 = [alpha(2:N) alpha(1)];
alphap1 = [alpha(N) alpha(1:N-1)];
betam1 = [beta(2:N) beta(1)];
betap1 = [beta(N) beta(1:N-1)];

a = betam1;
b = -alpha - 2*beta - 2*betam1;
c = alpha + alphap1 +betam1 + 4*beta + betap1;
d = -alphap1 - 2*beta - 2*betap1;
e = betap1;

% genera la matriz pentadiagonal
A = diag(a(1:N-2),-2) + diag(a(N-1:N),N-2);
A = A + diag(b(1:N-1),-1) + diag(b(N), N-1);
A = A + diag(c);
A = A + diag(d(1:N-1),1) + diag(d(N),-(N-1));
A = A + diag(e(1:N-2),2) + diag(e(N-1:N),-(N-2));

invAI = inv(A + gamma * diag(ones(1,N)));

for count = 1:ITER,
  vfx = interp2(fx,x,y,'*linear');
  vfy = interp2(fy,x,y,'*linear');

  pfx=interp2(px3,x,y,'*linear');
  pfy=interp2(py3,x,y,'*linear');

  temp_x=x;
  temp_y=y;

  % se agrega la fuerza de presion (un vector normal)
  xp = [x(2:N);x(1)];    yp = [y(2:N);y(1)];
  xm = [x(N);x(1:N-1)];  ym = [y(N);y(1:N-1)];

  qx = xp-xm;  qy = yp-ym;
  pmag = sqrt(qx.*qx+qy.*qy);
  px = qy./pmag;     py = -qx./pmag;

  % deforma el modelo activo
  x = invAI * (gamma* x + kappa*vfx + kappap.*px + 1*pfx);
  y = invAI * (gamma* y + kappa*vfy + kappap.*py + 1*pfy);

  x2=abs(temp_x-x);
  y2=abs(temp_y-y);
  x2=x2(:);
  y2=y2(:);
end
```

E.2 Implementación de los modelos

En esta sección se presenta la implementación de los tres modelos (Kass, Cohen y Xu) para los siguientes casos:

1. Imágenes formadas solamente de una línea negra y un fondo blanco. (Imágenes sencillas)
2. Casos de experimentación utilizando las imágenes del caso 1.
3. Imágenes con escala de grises. (Imágenes reales)

E.2.1 Código modelo Kass para imágenes sencillas.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Funcionamiento del contorno activo utilizando el modelo de Kass
%   aplicado a una imagen que consta de una linea negra sobre una
%   superficie blanca.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Se utilizan algoritmos propuestos por Chenyang Xu y Jerry L. Prince
%   otros son diseño de Miguel Angel Torres Tello (funcion manual,
%   variaciones a la funcion snakedeform, snakedisp2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
I1=imread('C:\prueba4.gif');

I=im2double(I1);        %se hace una transformacion de unidades uint8 a double.

f=I;

%otra opcion para poder operar con la imagen
%acorde a la ayuda de MATLAB, es utilizar el
%siguiente proceso alternativo.
%I=double(I1)+1;
%f = 1 - I/255;

f0 = gaussianBlur(f,1);    %se realiza una convlucion con una mascarilla gaussiana

[px,py] = gradient(-f0);    %se obtiene el gradiente de la imagen

figure(1);
    subplot(121); imshow(f); title('\fontsize{14} Imagen original');
    subplot(122); quiver(px,py);
    axis('image', 'off', 'ij');
    title('\fontsize{14} Vectores de fuerza externa');

% seleccion manual de puntos alrededor del contorno
[x,y]=manual(I);
figure(1); subplot(121);
snakedisp2(x,y,'r')

x2=x;
y2=y;       %variables utilizadas para medir el tiempo

%se realizan las iteraciones para mover los puntos
%de control hacia el contorno.

for i=1:10,
      [x,y] = snakedeform(x,y,0.05,0,1,2.15,px,py,5);
      [x,y] = snakeinterp(x,y,3,1);
      snakedisp2(x,y,'r')
      title(['Iteracion,  iter = ' num2str(i*5)])
      pause(0.25);
 end

%se realiza la misma operacion para poder tomar el tiempo
%de retardo de las funciones.

tic
for i=1:10,
      [x2,y2] = snakedeform(x2,y2,0.05,0,1,2.15,px,py,5);
      [x2,y2] = snakeinterp(x2,y2,3,1);
 end
toc;
```

```
t2=toc;
subplot(121);
imshow(f);
title(['Resultado final, tiempo = ', num2str(t2) ])
snakedisp2(x,y,'r')
```

E.2.2 Código modelo Cohen para imágenes sencillas.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Funcionamiento del contorno activo utilizando el modelo de Cohen
%   aplicado a una imagen que consta de una linea negra sobre una
%   superficie blanca.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Se utilizan algoritmos propuestos por Chenyang Xu y Jerry L. Prince
%   y Miguel Angel Torres Tello.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


I1=imread('C:\prueba4.gif');
%I1=imread('C:\error\4.gif');

I=im2double(I1);        %se hace una transformacion de unidades uint8 a double.
f=I;

%otra opcion para poder operar con la imagen
%acorde a la ayuda de MATLAB, es utilizar el
%siguiente proceso alternativo.
%I=double(I1)+1;
%f = 1 - I/255;

f0 = gaussianBlur(f,1);    %se realiza una convlucion con una mascarilla gaussiana

[px,py] = gradient(-f0);    %se obtiene el gradiente de la imagen

figure(1);
    subplot(121); imshow(f); title('\fontsize{14} Imagen original');
    subplot(122); quiver(px,py);
    axis('image', 'off', 'ij');
    title('\fontsize{14} Vectores de fuerza externa');

% seleccion manual de puntos alrededor del contorno
[x,y]=manual(I1);
figure(1); subplot(121);
snakedisp2(x,y,'r')

x2=x;
y2=y;       %variables utilizadas para medir el tiempo

%se realizan las iteraciones para mover los puntos
%de control hacia el contorno.

for i=1:10,
      [x,y] = snakedeform2(x,y,0.05,0,1,2.15,0.15,px,py,5);
      [x,y] = snakeinterp(x,y,3,1);
      snakedisp2(x,y,'r')
      title(['Iteracion,  iter = ' num2str(i*5)])
      pause(0.25);
 end

%se realiza la misma operacion para poder tomar el tiempo
%de retardo de las funciones.
```

```
tic
for i=1:10,
      [x2,y2] = snakedeform2(x2,y2,0.05,0,1,2.15,0.15,px,py,5);
      [x2,y2] = snakeinterp(x2,y2,3,1);
 end
toc;

t2=toc;
subplot(121);
imshow(f);
title(['Resultado final, tiempo = ', num2str(t2) ])
snakedisp2(x,y,'r')
```

E.2.3 Código modelo Xu para imágenes sencillas.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Funcionamiento del contorno activo utilizando el modelo de Xu
%   aplicado a una imagen que consta de una linea negra sobre una
%   superficie blanca.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Se utilizan algoritmos propuestos por Chenyang Xu y Jerry L. Prince
%   otros son diseño de Miguel Angel Torres Tello.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

I1=imread('C:\prueba4.gif');
%I1=imread('C:\error\4.gif');
I=im2double(I1);
%I=imnoise(I,'gaussian');
%I=imnoise(I,'salt & pepper',.05);
[row,columns]=size(I);

for i=1:columns,
   uno=find(I(:,i)>.4);
   I(uno,i)=1;
   cero=find(I(:,i)<=.4);
   I(cero,i)=0;
end


f=I;

    [u,v] = GVF(-f, 0.2, 80);
    mag = sqrt(u.*u+v.*v);
    px = u./(mag+1e-10); py = v./(mag+1e-10);

figure(1);
    quiver(px,py);
    axis off; axis equal; axis 'ij';
    title('\fontsize{14} Campo GVF');

    [x,y]=manual(I);

    %axis('square', 'off');
    %t = 0:0.05:6.28;
    %x = 32 + 25*cos(t);
    %y = 32 + 25*sin(t);
    %[x,y] = snakeinterp(x,y,2,.5);

    snakedisp2(x,y,'r')
```

```
    x4=x;
    y4=y;


figure(2);
    imshow(I);

  for i=1:25,
     [x4,y4] = snakedeform(x4,y4,0.05,0,1,0.6,px,py,5);
     [x4,y4] = snakeinterp(x4,y4,2,.5);
     snakedisp2(x4,y4,'r')
     title(['Iteracion,  iter = ' num2str(i*5)])
     pause(0.5);
  end

title('\fontsize{14} Seguimiento con el modelo de Xu');
snakedisp2(x,y,'r')
snakedisp2(x4,y4,'b')


figure(3);
imshow(I)
title('\fontsize{14} Resultado final con el modelo de Xu');
snakedisp2(x,y,'r')
snakedisp2(x4,y4,'b')
```

E.2.4 Código modelo Kass para experimentación con imágenes sencillas.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Experimentos
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Se utilizan algoritmos propuestos por Chenyang Xu y Jerry L. Prince
%   otros son diseño de Miguel Angel Torres Tello.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
I1=imread('C:\prueba4.gif');
%I1=imread('F:\error\4.gif');
I=im2double(I1);
%I=imnoise(I,'gaussian');
%I=imnoise(I,'salt & pepper',.05);
[row,columns]=size(I);

for i=1:columns,
   uno=find(I(:,i)>.5);
   I(uno,i)=1;
   cero=find(I(:,i)<=.5);
   I(cero,i)=0;
end

f=I;

    %para el modelo original
    f0 = gaussianBlur(I,1);
    [px2,py2] = gradient(-f0);

    %[x,y]=manual(I);

    axis('square', 'off');
    t = 0:0.05:6.28;
    x = 32 + 17*sin(t);
    %x= 91 + 83*sin(t);    %para imagen error/4.gif
```

```
    y = 35 + 15*cos(t);
    %y= 89 + 83*cos(t);    %para imagen error/4.gif
    [x,y] = snakeinterp(x,y,2,.5);



    contador1=1;
    contador2=1;
    k=0;
    j=0;
    fuer=0;

for fuer=.15:1:2.15,  %fuer=.15:.5:1.15

   for j=0.05:.25:.55,


      for k=0:.5:1,


         x2=x;
         y2=y;


         %Kass

            for i=1:50,
                [x3,y3,xtemp,ytemp] = snakedeform_ex(x3,y3,j,k,1,fuer,px2,py2,5);   %2
                [x3,y3] = snakeinterp(x3,y3,2,0.5);

                temp_x=max(xtemp);
                temp_y=max(ytemp);

                if ((temp_x<=.05)&&(temp_y<=.05))
                   break;

                end;
            end

         %figure(3);
         %imshow(I);
         %title('\fontsize{14} Seguimiento con el modelo de Kass');
         %snakedisp2(x,y,'r')
         %snakedisp2(x3,y3,'b')

         tempo=figura_temporal(I,x3,y3);
         %figure, imshow(tempo);

         %*******************************
         [contad,n]=porcentaje(I,tempo);
         final_Kass(contador1,contador2)=(contad/n)*100;

         contador2=contador2+1;
         contador3=contador2;

      end
      contador2=1;
      contador1=contador1+1;
   end
end
```

E.2.5 Código modelo Cohen para experimentación con imágenes sencillas.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Experimentos
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Se utilizan algoritmos propuestos por Chenyang Xu y Jerry L. Prince
%   otros son diseño de Miguel Angel Torres Tello.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
%I1=imread('F:\prueba4.gif');
I1=imread('F:\error\2.gif');
I=im2double(I1);
%I=imnoise(I,'gaussian');
%I=imnoise(I,'salt & pepper',.05);
[row,columns]=size(I);

for i=1:columns,
   uno=find(I(:,i)>.5);
   I(uno,i)=1;
   cero=find(I(:,i)<=.5);
   I(cero,i)=0;
end

f=I;

    %para el modelo original
    f0 = gaussianBlur(I,1);
    [px2,py2] = gradient(-f0);


    %[x,y]=manual(I);

    axis('square', 'off');
    t = 0:0.05:6.28;
    %x = 32 + 17*sin(t);
    x= 91 + 85*sin(t);    %para imagen error/4.gif
    %y = 35 + 15*cos(t);
    y= 89 + 85*cos(t);    %para imagen error/4.gif
    [x,y] = snakeinterp(x,y,2,.5);

    contador1=1;
    contador2=1;
    k=0;
    j=0;
    fuer=0;

for fuer=.15:1:2.15,  %fuer=.15:.5:1.15

   for j=0.05:.25:.55,


      for k=0:.5:1,


         x2=x;
         y2=y;

         %L.COHEN

            for i=1:50,
                 [x2,y2,xtemp,ytemp]  =  snakedeform2_ex(x2,y2,j,k,1,2.15,.15,px2,py2,5);  %kappap  .15  !!  variar
solo a 1.15 y 2.15
                 [x2,y2] = snakeinterp(x2,y2,2,.5);
```

```
                temp_x=max(xtemp);
                temp_y=max(ytemp);

                if ((temp_x<=.05)&&(temp_y<=.05))

                    break;

                end;
            end

        %figure(2);
        %imshow(I);
        %title('\fontsize{14} Seguimiento con el modelo de L.Cohen');
        %snakedisp2(x,y,'r')
        %snakedisp2(x2,y2,'b')

        [x2,y2,xtemp,ytemp] = snakedeform_ex(x2,y2,.05,0,1,2.15,px2,py2,5);   %2
        [x2,y2] = snakeinterp(x2,y2,2,0.5);

        tempo=figura_temporal(I,x2,y2);
        %figure, imshow(tempo);

        %********************************
        [contad,n]=porcentaje(I,tempo);
        final_Cohen(contador1,contador2)=(contad/n)*100;


        contador2=contador2+1;
        contador3=contador2;

      end
      contador2=1;
      contador1=contador1+1;
   end
end
```

E.2.6 Código modelo Xu para experimentación con imágenes sencillas.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Experimentos
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%
%   Se utilizan algoritmos propuestos por Chenyang Xu y Jerry L. Prince
%   otros son diseño de Miguel Angel Torres Tello.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
I1=imread('F:\prueba4.gif');
%I1=imread('F:\error\2.gif');
I=im2double(I1);
%I=imnoise(I,'gaussian');
%I=imnoise(I,'salt & pepper',.05);
[row,columns]=size(I);

for i=1:columns,
   uno=find(I(:,i)>.5);
   I(uno,i)=1;
   cero=find(I(:,i)<=.5);
   I(cero,i)=0;
end
```

```
f=I;


    %para el modelo original
    f0 = gaussianBlur(I,1);
    [px2,py2] = gradient(-f0);

    [u,v] = GVF(-f, 0.2, 80);
    mag = sqrt(u.*u+v.*v);
    px = u./(mag+1e-10); py = v./(mag+1e-10);


    %[x,y]=manual(I);

    axis('square', 'off');
    t = 0:0.05:6.28;
    x = 32 + 17*sin(t);
    %x= 91 + 85*sin(t);    %para imagen error/4.gif
    y = 35 + 15*cos(t);
    %y= 89 + 85*cos(t);    %para imagen error/4.gif
    [x,y] = snakeinterp(x,y,2,.5);



    contador1=1;
    contador2=1;
    k=0;
    j=0;
    fuer=0;

for fuer=.15:1:2.15,  %fuer=.15:.5:1.15

   for j=0.05:.25:.55,


      for k=0:.5:1,


         x4=x;
         y4=y;

         %Xu

           for i=1:50,
              [x4,y4,xtemp,ytemp] = snakedeform_ex(x4,y4,j,k,1,fuer,px,py,5); %.6
              [x4,y4] = snakeinterp(x4,y4,2,.5);

              temp_x=max(xtemp);
              temp_y=max(ytemp);

              if ((temp_x<=.05)&&(temp_y<=.05))
                break;

              end;
           end


         %figure(1);
         %imshow(I)
         %title('\fontsize{14} Resultado final con el modelo de Xu');
         %snakedisp2(x,y,'r')
         %snakedisp2(x4,y4,'b')

         [x4,y4,xtemp,ytemp] = snakedeform_ex(x4,y4,.05,.5,1,2,px2,py2,5);   %2
```

```
        [x4,y4] = snakeinterp(x4,y4,2,0.5);

        tempo=figura_temporal(I,x4,y4);
        %figure, imshow(tempo);

       %*******************************
        [contad,n]=porcentaje(I,tempo);
        final_Xu(contador1,contador2)=(contad/n)*100;

        contador2=contador2+1;
        contador3=contador2;

      end
      contador2=1;
      contador1=contador1+1;
   end
end
```

E.2.7 Código modelo Cohen-Kass para imágenes reales

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Funcionamiento del contorno activo utilizando el modelo de Cohen
%   aplicado a una imagen real
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Se utilizan algoritmos propuestos por Chenyang Xu y Jerry L. Prince
%   otros son diseñode Miguel Angel Torres Tello.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
I1=imread('F:\imagenes\prueba12.png');

I_bin=im2double(I1);        %se hace una transformacion de unidades uint8 a double.

[row,columns]=size(I_bin);
for i=1:columns,
   uno=find(I_bin(:,i)>.5);
   I_bin(uno,i)=1;
   cero=find(I_bin(:,i)<=.5);
   I_bin(cero,i)=0;
end

%otra opcion para poder operar con la imagen
%acorde a la ayuda de MATLAB, es utilizar el
%siguiente proceso alternativo.
%I=double(I1)+1;
%f = 1 - I/255;

%f0 = gaussianBlur(f,1);    %se realiza una convlucion con una mascarilla gaussiana

%[px,py] = gradient(-f0);    %se obtiene el gradiente de la imagen


%******************
I=im2double(I1);
J=histeq(I);
[px3,py3]=energy_term(J);
%J=imadjust(I,[.5 .6],[]);
I2=gaussianBlur(I,1);
% Obtencion de la energia externa
[Ix,Iy]=gradient(I2);
px=gradient(abs(Ix).^(1/2));
```

```
Iy=permute(Iy,[2 1]);
py=gradient(abs(Iy).^(1/2));
py=ipermute(py,[2 1]);
%******************

f=I;
figure(1);
    subplot(121); imshow(f); title('\fontsize{14} Imagen original');
    subplot(122); quiver(px,py);
    axis('image', 'off', 'ij');
    title('\fontsize{14} Vectores de fuerza externa');

% seleccion manual de puntos alrededor del contorno
%[x,y]=manual(I);

    %axis('square', 'off');
    %t = 0:0.05:6.28;
    %x = 260 + 210*cos(t);  %para prueba16
    %y = 250 + 175*sin(t);  %para prueba16

    %********
    %modelos de prueba para prueba12
    x=[135.0256  133.0963  165.4125  314.4532  310.5945  271.5256  252.7147  195.3171  152.8719];
    y=[75.9850  166.1811  219.7200  170.0398   99.1369   37.8807   32.5751   55.2447   70.6793];
    %********
    [x,y] = snakeinterp(x,y,2,.5);

   figure(1); subplot(121);
snakedisp2(x,y,'r')

x2=x;
y2=y;      %variables utilizadas para medir el tiempo


%se realizan las iteraciones para mover los puntos
%de control hacia el contorno.

for i=1:50,
     [x,y,xtemp,ytemp] = snakedeform_ex2(x,y,0.30,0,1,2.15,.15,px,py,5,px3,py3); %2.15 external force
     [x,y] = snakeinterp(x,y,2,.5);

     temp_x=max(xtemp);
     temp_y=max(ytemp);

     if ((temp_x<=.2)&&(temp_y<=.2))
        'cumplio'
        break;
     end

     snakedisp2(x,y,'r')
     title(['Iteracion,  iter = ' num2str(i*5)])
     pause(0.25);
 end

[x,y,xtemp,ytemp] = snakedeform_ex(x,y,.05,0,1,2.15,px,py,5);   %2
[x,y] = snakeinterp(x,y,2,0.5);

snakedisp2(x,y,'r')
     title(['Iteracion,  iter = ' num2str(i*5)])
     pause(0.25);



%se realiza la misma operacion para poder tomar el tiempo
```

```
%de retardo de las funciones.

tic
for i=1:50,
    [x2,y2,xtemp,ytemp] = snakedeform_ex2(x2,y2,0.3,0,1,2.15,.15,px,py,5,px3,py3);
    [x2,y2] = snakeinterp(x2,y2,2,.5);

    temp_x=max(xtemp);
    temp_y=max(ytemp);

    if ((temp_x<=.2)&&(temp_y<=.2))
       'cumplio'
       break;
    end
end
[x2,y2,xtemp,ytemp] = snakedeform_ex(x2,y2,.05,0,1,2.15,px,py,5);   %2
[x2,y2] = snakeinterp(x2,y2,2,0.5);
toc;

t2=toc;
subplot(121);
imshow(f);
title(['\fontsize{14} Resultado final, tiempo = ', num2str(t2) ])
snakedisp2(x,y,'r')
tempo=figura_temporal(I_bin,x,y);
[contad,n]=porcentaje2(I_bin,tempo);
(contad/n)*100
```