PTC5750 – Técnicas avançadas em processamento de imagens médicas

Monografia sobre segmentação de imagens 2D baseada em contornos ativos (*snakes*) paramétricos com *Gradient Vector Flow* (*GVF*)

Marcelo Monari Baccaro – 8989262

1) Introdução

O processamento digital de imagens é um campo com muitas subdivisões, entre estes, um dos mais complexos é a segmentação de imagens, muito utilizado na área médica para descrição das dimensões físicas de uma porção do corpo humano.

O objetivo da segmentação está em rotular os pixels com relação ao pertencimento deles a um determinado objeto de interesse. Uma estratégia para realizar isto é destacar as fronteiras, como a técnica de contornos ativos (*snakes*), que usa conceitos de equilíbrio de forças (internas e externas), energia potencial (de forças internas e externas) e calculo variacional (como na mecânica analítica) para descrever equações evolutivas que buscam estabilizar uma linha contínua nas fronteiras.

Nesta monografia, será descrito o método de segmentação variacional de imagens bidimensionais por contorno ativo paramétrico com fluxo de vetor gradiente (*Gradient Vector Flow - GVF*). Um contorno ativo paramétrico usa uma curva paramétrica $\gamma(\boldsymbol{x}(s)) \in \mathbb{R}^2$, $\boldsymbol{x}(s) = (x(s), y(s)) \in \mathbb{R}^2$, $x, y \in \mathbb{R}$, $s \in [0,1]$, para descrever as fronteiras do objeto, sendo que ela possui propriedades mecânicas como elasticidade (α) e rigidez (β). O fluxo de vetor gradiente representa um campo vetorial de forças externas $\boldsymbol{F}_{ext}(x,y) = (u(x,y), v(x,y)) \in \mathbb{R}^{2\cdot(MxN)}$, $u, v \in \mathbb{R}^{MxN}$ gerado a partir da intensidade dos pixels da imagem original $I(x,y) \in \mathbb{R}^{MxN}$.

2) Método

O método de segmentação por *GVF snakes* foi introduzido por Chenyang Xu e Jerry L. Prince em [1]. Esta monografia basicamente descreverá esta técnica, além de apresentar a sua aplicação numérica.

A técnica começa com a definição de funcionais para fazer o papel da energia potencial, em que as forças internas elásticas são representadas pela energia potencial elástica, enquanto o gradiente da energia potencial externa forma as forças externas. Em seguida é aplicada a Equação de Euler para encontrar um sistema de equações diferenciais que representam a solução ótima sem restrição. Entretanto, resolver este sistema de equações não é fácil, então usa-se a abordagem de equações evolutivas da programação dinâmica para resolvê-las iterativamente. Por fim, usando o Método das Diferenças Finitas chega-se a solução numérica.

2.1) Funcionais das energias potencias

2.1.1) Energia mecânica total do contorno ativo

$$E_{snake} = \int_0^1 \left[\frac{1}{2} \cdot (\alpha \cdot |\boldsymbol{x}'(s)|^2 + \beta \cdot |\boldsymbol{x}''(s)|^2) + E_{ext}(\boldsymbol{x}(s))\right] \cdot ds$$

2.1.2) Energia potencial externa

$$E_{ext}(\boldsymbol{x}(s)) = \int\int \left[\mu \cdot \left(u_x^2 + u_y^2 + v_x^2 + v_y^2\right) + |\nabla f|^2 \cdot |\boldsymbol{F}_{ext} - \nabla f|^2\right] \cdot dx \cdot dy$$

Sendo que,

$$f = -|\nabla G_\sigma(x,y) * I(x,y)|^2, em\ que\ G_\sigma(x,y) = exp\left(-\frac{x^2+y^2}{2\cdot\sigma^2}\right)$$

2.2) Aplicação da Equação de Euler

A Equação de Euler busca minimizar o funcional:

$$J(c) = \int_a^b F\big(s, c(s), c'(s)\big) \cdot ds$$

Através da equação diferencial parcial:

$$\frac{\partial J}{\partial c} = \frac{\partial F}{\partial c} - \frac{d}{ds}\left(\frac{\partial F}{\partial c'}\right) = 0$$

2.2.1) Equação da dinâmica do contorno ativo $\boldsymbol{x}(s) = \big(x(s), y(s)\big)$

$$\alpha \cdot \boldsymbol{x}''(s) - \beta \cdot \boldsymbol{x}''''(s) + \boldsymbol{F}_{ext}(x,y) = 0, sendo\ que\ \boldsymbol{F}_{ext}(x,y) = -\nabla E_{ext}(x,y)$$

2.2.2) Equação da força de campo externo $\boldsymbol{F}_{ext} = \big(u(x,y), v(x,y)\big)$

$$\mu \cdot \nabla^2 u(x,y) - \big(u(x,y) - f_x(x,y)\big) \cdot \left(f_x^2(x,y) + f_y^2(x,y)\right) = 0$$

$$\mu \cdot \nabla^2 v(x,y) - \left(v(x,y) - f_y(x,y)\right) \cdot \left(f_x^2(x,y) + f_y^2(x,y)\right) = 0$$

2.3) Equações evolutivas da programação dinâmica

Como calcular as equações diferenciais acima não é nem um pouco trivial. Usa-se a programação dinâmica com o método do gradiente (*Gradient descent*) para chegar nas equações diferenciais parciais evolutivas.

$$\frac{\partial c}{\partial t} = -\frac{\partial J}{\partial c} = -\frac{\partial F}{\partial c} + \frac{d}{ds}\left(\frac{\partial F}{\partial c'}\right)$$

2.3.1) Equação da dinâmica do contorno ativo $\boldsymbol{x}(s,t) = \big(x(s,t), y(s,t)\big)$

$$\frac{\partial \boldsymbol{x}}{\partial t}(s,t) = \alpha \cdot \frac{\partial^2 \boldsymbol{x}}{\partial s^2}(s,t) - \beta \cdot \frac{\partial^4 \boldsymbol{x}}{\partial s^4}(s,t) + \boldsymbol{F}_{ext}(x,y)$$

2.3.2) Equação da força de campo externo $\boldsymbol{F}_{ext}(x,y,t) = \big(u(x,y,t), v(x,y,t)\big)$

$$u_t(x,y,t) = \mu \cdot \nabla^2 u(x,y,t) - \big(u(x,y,t) - f_x(x,y)\big) \cdot \Big(f_x^2(x,y) + f_y^2(x,y)\Big)$$

$$v_t(x,y,t) = \mu \cdot \nabla^2 v(x,y,t) - \Big(v(x,y,t) - f_y(x,y)\Big) \cdot \Big(f_x^2(x,y) + f_y^2(x,y)\Big)$$

2.4) Solução numérica pelo Método das Diferenças finitas

As simulações numéricas foram feitas em Matlab. Os códigos estão presentes no Apêndice ao final da monografia.

2.4.1) Equação da dinâmica discreta do contorno ativo (*snake_deform.m*)

Aproximação temporal por diferenças finitas progressivas:

$$x_t(s,t) \approx \frac{x_m^{n+1} - x_m^n}{\Delta T} \qquad y_t(s,t) \approx \frac{y_m^{n+1} - y_m^n}{\Delta T}$$

Aproximação espacial por diferenças finitas centradas:

$$x_{ss}(s,t) \approx \frac{x_{m-1}^n - 2x_m^n + x_{m+1}^n}{\Delta S^2} \qquad y_{ss}(s,t) \approx \frac{y_{m-1}^n - 2y_m^n + y_{m+1}^n}{\Delta S^2}$$

$$x_{ssss}(s,t) \approx \frac{x_{m-2}^n - 4x_{m-1}^n + 6x_m^n - 4x_{m+1}^n + x_{m+2}^n}{\Delta S^4}$$

$$y_{ssss}(s,t) \approx \frac{y_{m-2}^n - 4y_{m-1}^n + 6y_m^n - 4y_{m+1}^n + y_{m+2}^n}{\Delta S^4}$$

Juntando os dois conjuntos de equações diferenças finitas para $\Delta S = 1$:

$$x_m^{n+1} = x_m^n + \Delta T(-\beta x_{m-2}^n + (\alpha + 4\beta)x_{m-1}^n - (2\alpha + 6\beta)x_m^n + (\alpha + 4\beta)x_{m+1}^n - \beta x_{m+2}^n) + u(x_m^n, y_m^n)$$

$$y_m^{n+1} = y_m^n + \Delta T(-\beta y_{m-2}^n + (\alpha + 4\beta)y_{m-1}^n - (2\alpha + 6\beta)y_m^n + (\alpha + 4\beta)3y_{m+1}^n - \beta y_{m+2}^n) + v(x_m^n, y_m^n)$$

Para fazer com que a aproximação numérica da derivada da curva paramétrica $\gamma\big(\boldsymbol{x}(s)\big)$ em relação ao parâmetro, ou seja, $\gamma'\big(\boldsymbol{x}(s)\big)$, permaneça próxima da real, foi necessário usar uma função de interpolação (*snake_interp.m*), que adiciona ou remove pontos de acordo com a definição de uma distância mínima entre os ponto (d_{min}, para que não haja acumulo de pontos, o que diminui a carga do

processamento numérico) e uma máxima (d_{max}, para que não haja uma grande divergência da derivada). Embora isso resolva o problema da aproximação da derivada, a quantidade de pontos do contorno ativo pode variar a cada, o que me obriga a usar um método numérico explicito, além de usar a variação do funcional de energia como critério de parada.

O contorno ativo é inicializado como um círculo cujo diâmetro é próximo das menor das dimensões da imagem (*snake_init.m*). E o campo de força é interpolado linearmente para os pontos que estão entre os pontos da grade deste.

2.4.2) Força de campo discreto externo (*gvf.m*)

Aproximação temporal por diferenças finitas progressivas:

$$u_t(x,y,t) \approx \frac{u_{i,j}^{k+1} - u_{i,j}^{k}}{\Delta t} \qquad v_t(x,y,t) \approx \frac{v_{i,j}^{k+1} - v_{i,j}^{k}}{\Delta t}$$

Aproximação espacial por diferenças finitas centradas:

$$\nabla^2 u(x,y,t) \approx \frac{u_{i+1,j}^{k} + u_{i-1,j}^{k} - 4u_{i,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k}}{\Delta x \cdot \Delta y}$$

$$\nabla^2 v(x,y,t) \approx \frac{v_{i+1,j}^{k} + v_{i-1,j}^{k} - 4v_{i,j}^{k} + v_{i,j+1}^{k} + v_{i,j-1}^{k}}{\Delta x \cdot \Delta y}$$

Aplicando estes dois conjuntos de equações de diferenças na equação diferencial do GVF com $r = (\mu \cdot \Delta t)/(\Delta x \cdot \Delta y)$ (condição CFL de estabilidade → $r \leq 1/4$):

$$u_{i,j}^{k+1} = \left(1 - b_{i,j} \cdot \Delta t\right) \cdot u_{i,j}^{k} + r \cdot \left(u_{i+1,j}^{k} + u_{i-1,j}^{k} - 4u_{i,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k}\right) + c_{i,j}^{1} \cdot \Delta t$$

$$v_{i,j}^{k+1} = \left(1 - b_{i,j} \cdot \Delta t\right) \cdot v_{i,j}^{k} + r \cdot \left(v_{i+1,j}^{k} + v_{i-1,j}^{k} - 4v_{i,j}^{k} + v_{i,j+1}^{k} + v_{i,j-1}^{k}\right) + c_{i,j}^{2} \cdot \Delta t$$

Sendo que,

$$b_{i,j} = \left(f_{x_{i,j}}\right)^2 + \left(f_{y_{i,j}}\right)^2$$

$$c_{i,j}^{1} = b_{i,j} \cdot f_{x_{i,j}}$$

$$c_{i,j}^{2} = b_{i,j} \cdot f_{y_{i,j}}$$

O campo de força do GVF também é resolvido de maneira iterativa assim como a equação dinâmica do contorno ativo, entretanto o critério de parada dele é feito através de comparação com o estado anterior. Ele é inicializado com $u_{i,j}^0 = f_{x\,i,j}$ e $v_{i,j}^0 = f_{y_{i,j}}$.

3) Resultados

O códigos que geram saídas são:

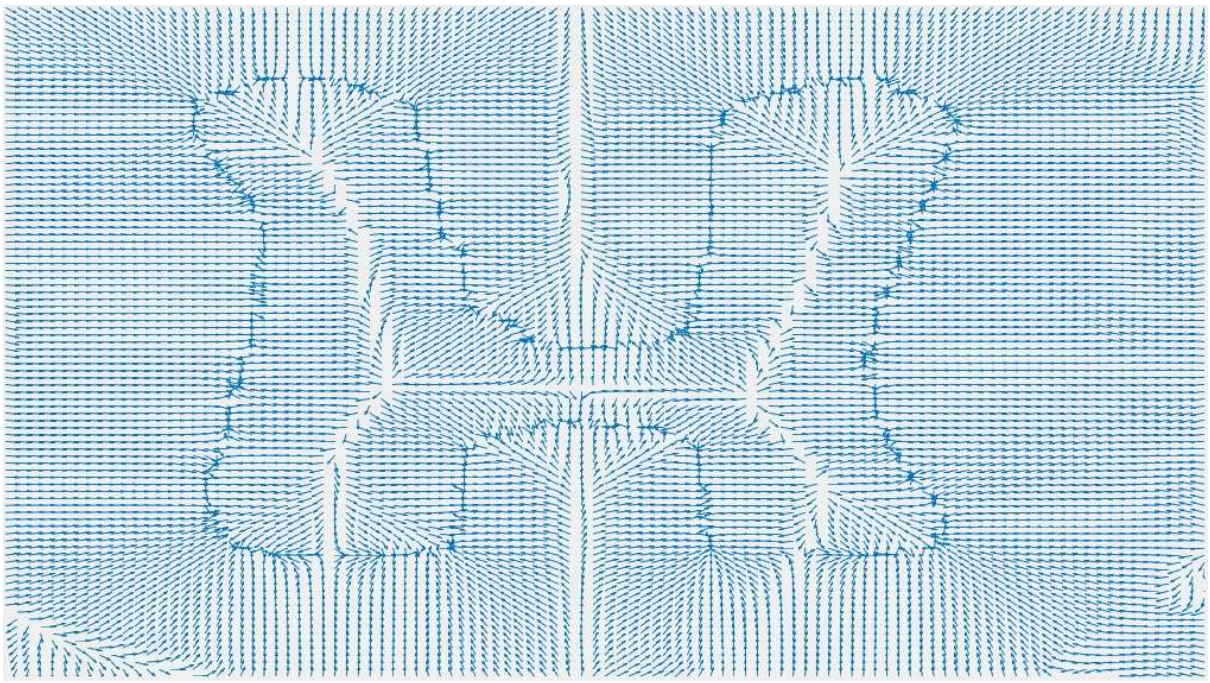
i) *segmentation_accuracy_measures.m* → Métricas de Verdadeiro Positivo (TP), Verdadeiro Negativo (TN) e Falso Positivo (FP)

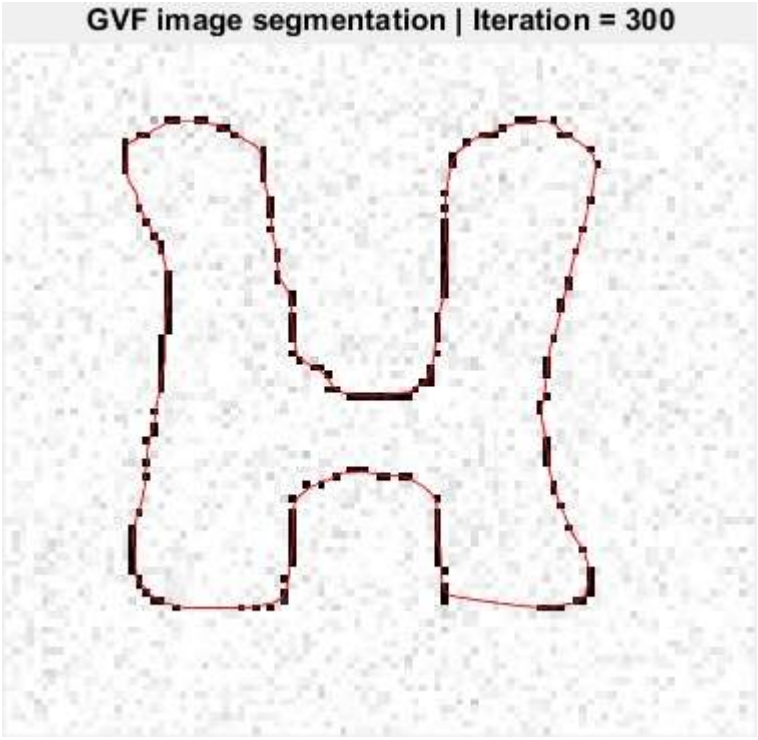
ii) *segmentation_image_output.m* → Converte as coordenadas do contorno ativo em valores discreto como nos pixels da imagem, gerando a imagem binária da segmentação

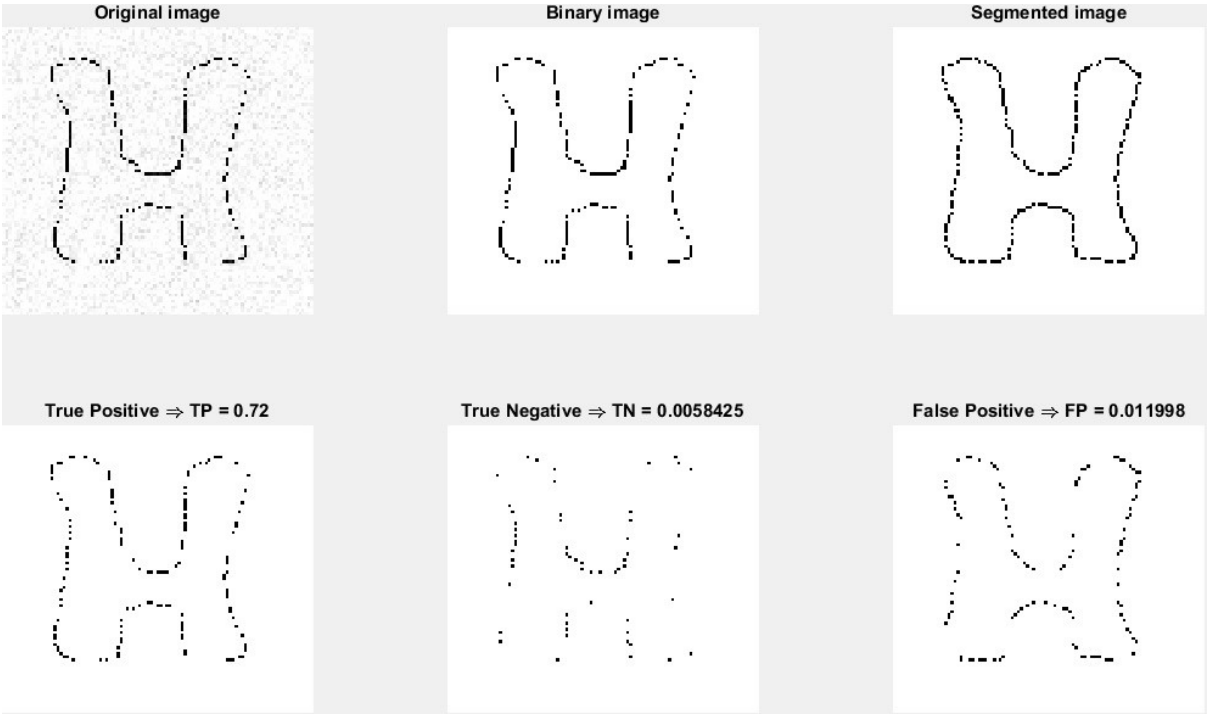
iii) *snake_print.m* → Imprime o contorno ativo por cima da imagem original

iv) *vector_field_print.m* → Imprime o campo de forças do GVF

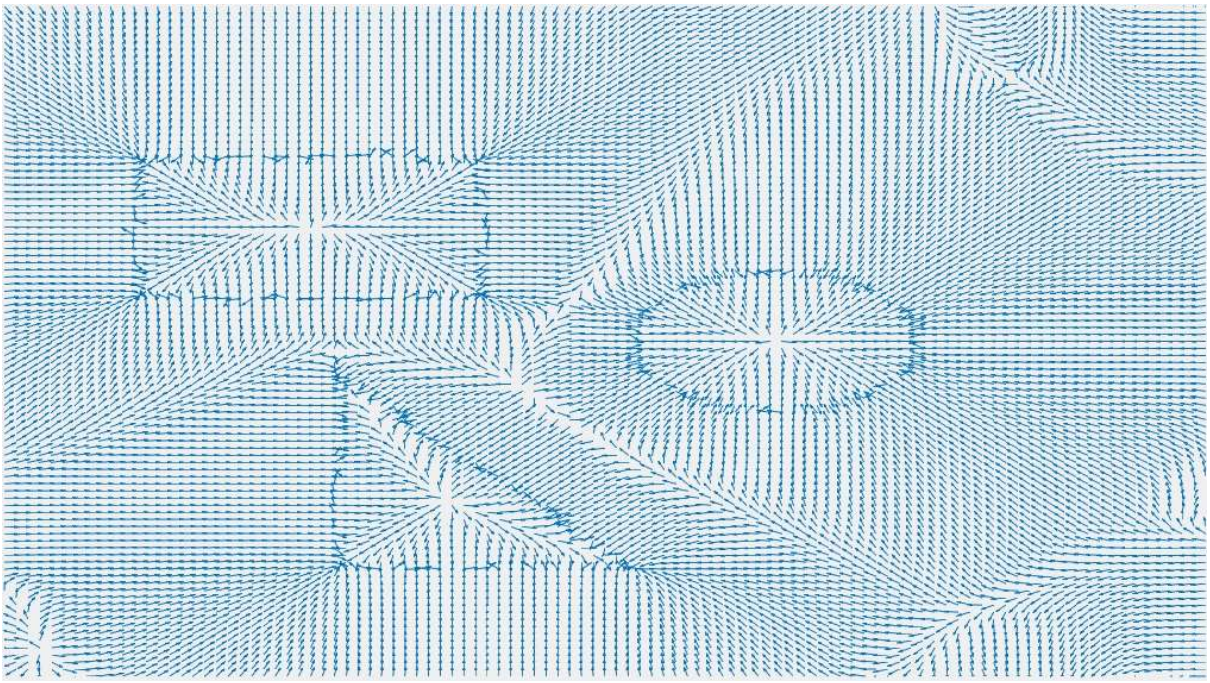
3.1) Figura de uma H corrompido com ruído gaussiano com 0.003 de variância:

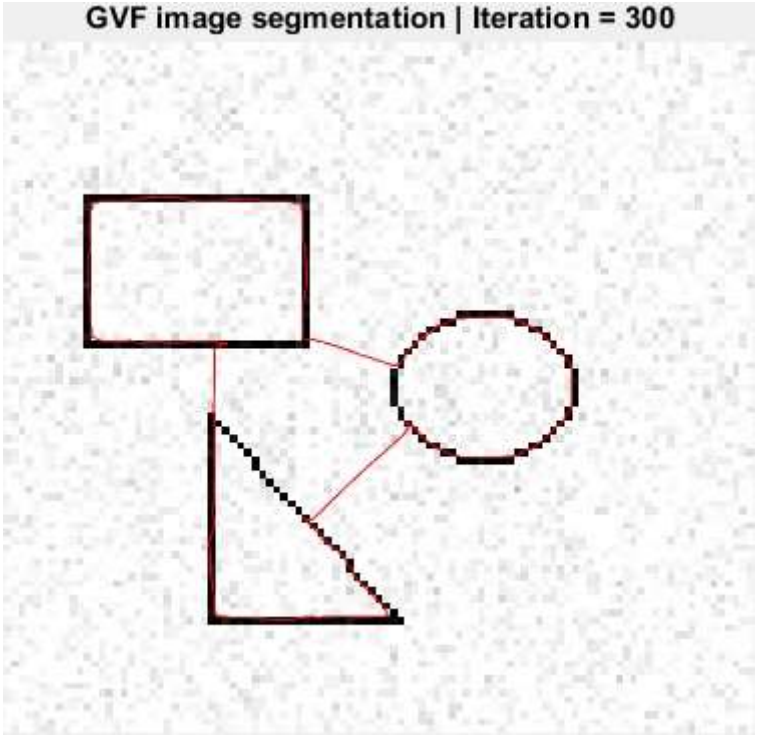


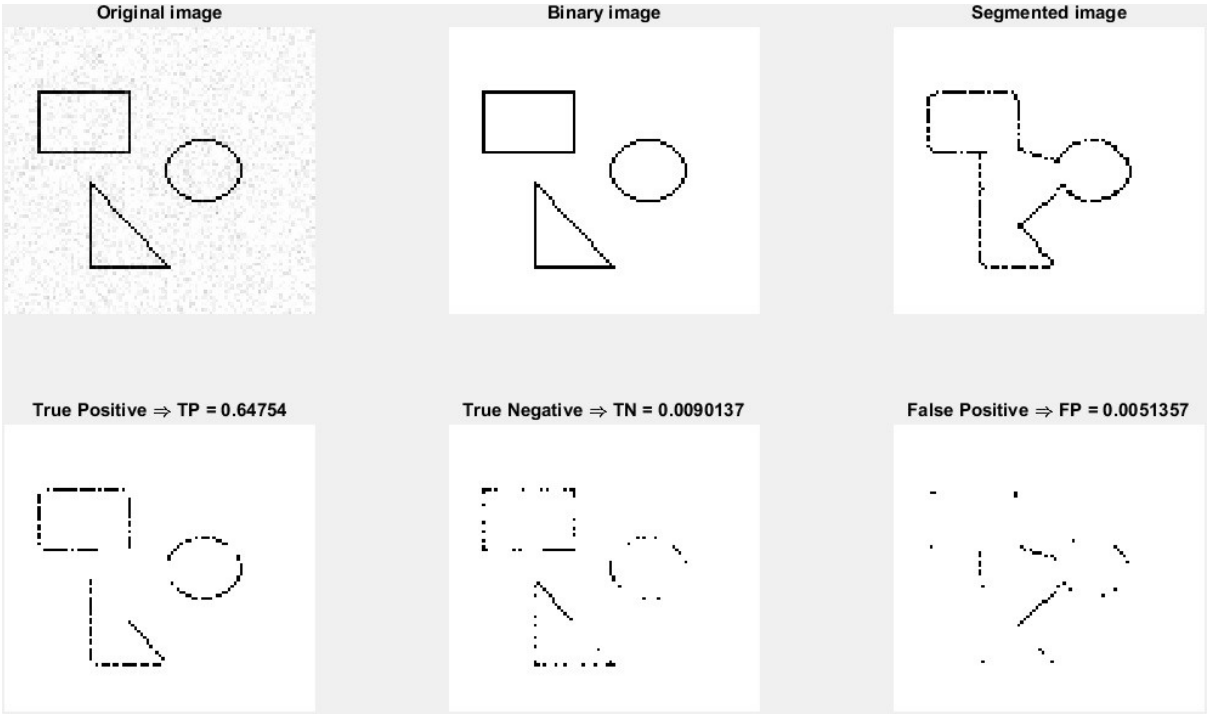




3.2) Figuras geométricas com ruído gaussiano com 0.003 de variância:




GVF image segmentation | Iteration = 300



4) Conclusão

A técnica de segmentação de imagens por contornos ativos paramétricos com GVF se mostrou capaz de se difundir mesmo em regiões em forma de “U”. E isto se deve a forma como o campo de forças é gerado, em que, para o caso contínuo e para uma região conexa, cada ponto dentro desta região é mapeado para um ponto diferente da fronteira desta região conexa. No caso discreto, ocorrem regiões cujos pontos acabam se difundindo para um mesmo ponto, gerando um ponto de acumulação. Mas, conforme a densidade de pontos aumenta (ou seja, a resolução), o caso discreto se aproxima do contínuo e a distribuição de pontos do contorno ativo se torna mais uniforme.

Embora este método seja muito bom para segmentar uma única fronteira, mesmo com um pouco de ruído, como se pode ver no segundo caso, para objetos com fronteiras não-conexas, este método é incapaz de realizar uma segmentação completa, pois o contorno ativo não se quebra.

É importante ressaltar que os exemplos de imagens usados para segmentação usados têm a linha de fronteira mais grossa do que apenas um único pixel, o que afeta na medida da acurácia. Ainda assim é possível ver qualitativamente que o método fez uma excelente segmentação no caso do “H”.

5) Referências

[1] Chenyang Xu et al. Snakes, Shapes, and Gradient Vector Flow. IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 7, NO. 3, MARCH 1998, 359-369

6) Apêndices

1) GVF_SNAKE.m

```
clear; clc; close;
I = imread('H1_gray.jpg');
%I = imread('GeometricShapes.jpg');
[M, N] = size(I);
I = imnoise(I, 'gaussian', 0, 0.003);
%I = imnoise(I, 'salt & pepper', 0.05);
%I = imnoise(I, 'speckle', 0.01);

%%% GRADIENT VECTOR FLOW
gvf_mu = 0.1;   gvf_dt = 1;   gvf_tol = 1e-3;
[u, v] = gvf (I, gvf_mu, gvf_dt, gvf_tol);
%vector_filed_print (u, v, N, M);

%%% SNAKE INIT
dmin = 0.5;   dmax = 2;
[x, y] = snake_init (M, N, dmin, dmax);
snake_print (I, x, y, 0);

%%% SNAKE DEFORMANTION
alpha = 0.1;   beta = 0.01;   dt = 0.6;   tol = 0.01;   it_max = 300;
[x, y, it] = gvf_snake_segmentation (I, x, y, u, v, alpha, beta, dmin,
dmax, dt, M, N, tol, it_max);

%%% OUTPUT IMAGES
Ib = im2bw(I, 0.2);
Is = segmentation_image_output (x, y, M, N);
[ITP, TP, ITN, TN, IFP, FP] = segmentation_accuracy_measures (Ib, Is, M,
N);
figure(2), set(gcf, 'Position', get(0, 'Screensize'));
subplot(2,3,1), imshow(I), title('Original image');
subplot(2,3,2), imshow(Ib), title('Binary image');
subplot(2,3,3), imshow(Is), title('Segmented image');
subplot(2,3,4), imshow(ITP), title(['True Positive \Rightarrow TP = '
num2str(TP)]);
subplot(2,3,5), imshow(ITN), title(['True Negative \Rightarrow TN = '
num2str(TN)]);
subplot(2,3,6), imshow(IFP), title(['False Positive \Rightarrow FP = '
num2str(FP)]);
```

2) gvf.m

```
function [u, v] = gvf (I, mu, dt, tol)
I = double(I);
%%% Edge map: f = (conv(I, laplacian(G)))^2
f = imfilter(I, fspecial('log', [11, 11], 0.2), 'symmetric') .^ 2;
fmin  = min(f(:));
fmax  = max(f(:));
f = (f - fmin) / (fmax - fmin);

%%% Gradient Vector Flow
fx = imfilter(f, -transpose(fspecial('sobel'))/4, 'symmetric');
fy = imfilter(f, -fspecial('sobel')/4, 'symmetric');

% Explicit method: CFL step-size restriction -> r <= 1/4
dx = 1;
```

```
dy = 1;
r = mu * dt / (dx * dy);
b = dt * (fx.^2 + fy.^2);
bn = 1 - b;
c1 = b .* fx;
c2 = b .* fy;
u = fx; % u initial condition
v = fy; % v initial condition
emax = 1; % Absolute error
lap = fspecial('laplacian', 0); % Laplacian filter
while emax > tol % error tolerance
    ub = u; % u before
    u = bn .* u + r * imfilter(u, lap, 'symmetric') + c1;
    vb = v; % v before
    v = bn .* v + r * imfilter(v, lap, 'symmetric') + c2;
    emax = max( max(max(abs(u-ub))) , max(max(abs(v-vb))) );
end
mag = sqrt(u.*u + v.*v);
u = u ./ (mag + 1e-10);
v = v ./ (mag + 1e-10);
```

3) vector_filed_print.m

```
function vector_filed_print (u, v, N, M)
u = flip(u);
v = -flip(v);

figure(3);
quiver(u, v);
axis off;
xlim([1,N]);
ylim([1,M]);
```

4) gvf_snake_segmentation.m

```
function [x, y, it] = gvf_snake_segmentation (I, x, y, u, v, alpha, beta,
dmin, dmax, dt, M, N, tol, it_max)
dE = 1; % energy relative difference
it = 0; % iteration
while (dE > tol / 100) && (it < it_max)
    [x, y, E] = snake_deform (x, y, u, v, alpha, beta, dt, M, N);
    [x, y] = snake_interp (x, y, dmin, dmax);
    if it > 0
        dE = abs(1 - Eb/E);
    end
    Eb = E;
    it = it + 1;
    snake_print (I, x, y, it);
end
```

5) snake_init.m

```
function [x, y] = snake_init (M, N, dmin, dmax)
cx = floor(N / 2); % image x center
cy = floor(M / 2); % image y center
rho = min(cx, cy) - 3; % radius
```

```
d = (dmax + dmin) / 2; % mean distance
as = acos(1 - (d/rho)^2 / 2); % angle step
ang = -(0 : as : 2*pi - as); % angles in clockwise

x = cx + rho * cos(ang); % x coord points
y = cy + rho * sin(ang); % y coord points
```

6) snake_deform.m

```
function [x, y, E] = snake_deform (x, y, u, v, alpha, beta, dt, M, N)
% alpha: elasticity parameter
% beta:  rigidity parameter
% gamma: viscosity parameter
% u & v: external force field

n = length(x);
u = interp2(u, x, y, 'linear');
v = interp2(v, x, y, 'linear');

x_b1 = x([n, 1:n-1]);      % x[n-1]
x_b2 = x([n-1, n, 1:n-2]); % x[n-2]
x_a1 = x([2:n, 1]);        % x[n+1]
x_a2 = x([3:n, 1, 2]);     % x[n+2]
dx2 = x_b1 - 2*x + x_a1; % ~ x''(s)
dx4 = x_b2 - 4*x_b1 + 6*x - 4*x_a1 + x_a2; % ~ x''''(s)
x = x + dt * (alpha*dx2 - beta*dx4 + u);

y_b1 = y([n, 1:n-1]);      % y[n-1]
y_b2 = y([n-1, n, 1:n-2]); % y[n-2]
y_a1 = y([2:n, 1]);        % y[n+1]
y_a2 = y([3:n, 1, 2]);     % y[n+2]
dy2 = y_b1 - 2*y + y_a1; % ~ y''(s)
dy4 = y_b2 - 4*y_b1 + 6*y - 4*y_a1 + y_a2; % ~ y''''(s)
y = y + dt * (alpha*dy2 - beta*dy4 + v);

% Image size limits
for i = 1 : n
    if x(i) < 1
        x(i) = 1;
    elseif x(i) > N
        x(i) = N;
    end
    if y(i) < 1
        y(i) = 1;
    elseif y(i) > M
        y(i) = M;
    end
end

% Energy
dx1 = x_a1 - x_b1; % ~ x'(s)
dy1 = y_a1 - y_b1; % ~ y'(s)
E = 0.5 * (alpha * sum(sqrt(dx1.^2 + dy1.^2)) + beta * sum(sqrt(dx2.^2 +
dy2.^2))) + sum(u) + sum(v);
```

7) snake_interp.m

```
function [xf, yf] = snake_interp (x, y, dmin, dmax)
```

```
% -> Mark points that are too close from each other for removal and
%    calculate how many interpolation points between points that are
%    too far away from each other
n = length(x);
xr = x(1); % x reference
yr = y(1); % y reference
idx_min = zeros(1, n); % Mark points that are too close
idx_max = zeros(1, n+1); % Number of points to add between points
count_min = 0; % Counts points to remove
for i = 2 : n
    d = ((x(i)-xr)^2 + (y(i)-yr)^2) ^ 0.5;
    if d < dmin
        idx_min(i) = 1;
        count_min = count_min + 1;
    else
        xr = x(i); yr = y(i); % Reference
        if d > dmax % Number of points to add between
            idx_max(i) = floor(d / dmax);
        end
    end
end
% Dealing with the end of the snake
d = ((x(1)-xr)^2 + (y(1)-yr)^2) ^ 0.5;
if d < dmin % Removal
    % Find the position of the current reference
    k = n;
    while idx_min(k) == 1
        k = k - 1;
    end
    % Mark it for removal
    idx_min(k) = 1;
    idx_max(k) = 0;
    count_min = count_min + 1;
    % Find the position of the previous reference
    k = k - 1;
    while idx_min(k) == 1
        k = k - 1;
    end
    % Mark how many interpolation points until to the end
    xr = x(k); yr = y(k); % reference
    d = ((x(1)-xr)^2 + (y(1)-yr)^2) ^ 0.5;
    idx_max(n+1) = floor(d / dmax);
elseif d > dmax % Interpolation
    idx_max(n+1) = floor(d / dmax);
end

% -> Remove points that are too close and interpolate points that are
%    too far away from each other
nf = n - count_min + sum(idx_max);
xf = zeros(1, nf); xf(1) = x(1); % x output
yf = zeros(1, nf); yf(1) = y(1); % y output
pos = 2; % Current posion of the output vectors
for i = 2 : n
    if idx_min(i) == 0 % This point isn't for removal
        nm = idx_max(i); % Number of points for interpolations
        if nm > 0 % LINEAR interpolation
            dx = (x(i) - xf(pos-1)) / (nm + 1); % x step
            dy = (y(i) - yf(pos-1)) / (nm + 1); % y step
            for j = (0 : nm-1) + pos % Fill with the interpolation
                xf(j) = xf(j-1) + dx;
                yf(j) = yf(j-1) + dy;
```

```
            end
            pos = pos + nm;
        end
        xf(pos) = x(i);
        yf(pos) = y(i);
        pos = pos + 1;
    end
end
% Dealing with the end of the snake
nm = idx_max(n+1);
if nm > 0 % linear interpolation
    dx = (x(1) - xf(pos-1)) / (nm + 1);
    dy = (y(1) - yf(pos-1)) / (nm + 1);
    for j = (0 : nm-1) + pos
        xf(j) = xf(j-1) + dx;
        yf(j) = yf(j-1) + dy;
    end
end
```

8) snake_print.m

```
function snake_print (I, x, y, i)
imshow(I);

if i == 0
    set(gcf, 'Position', get(0, 'Screensize'));
end

hold on;
plot([x, x(1)], [y, y(1)], 'r');
hold off;

title(['GVF image segmentation | Iteration = ' num2str(i)]);

pause(0.001);
```

9) segmentation_image_output.m

```
function I_seg = segmentation_image_output (x, y, M, N)
xr = round(x);
yr = round(y);
I_seg = 255 * ones(M, N);
for i = 1 : length(xr)
        I_seg(yr(i), xr(i)) = 0;
end
I_seg = im2bw(I_seg, 0.5);
```

10) segmentation_accuracy_measures.m

```
function [ITP, TP, ITN, TN, IFP, FP] = segmentation_accuracy_measures (Ib,
Is, M, N)
cp = sum(sum(~Ib));
cn = M * N - cp;

% True Positive
```

```
ITP = Ib | Is;
TP = sum(sum(~ITP)) / cp;

% True Negative
ITN = (~xor(Ib, Is)) | Ib;
TN = sum(sum(~ITN)) / cn;

% False Positive
IFP = (~xor(Ib, Is)) | Is;
FP = sum(sum(~IFP)) / cn;
```