به نام خدا

مینی‌پروژه ۵

مرز فعال هندسی تطبیق بایرنری محلی

Active Contours Driven by Local Binary Fitting Energy

(LBF)

سید سجاد ائمی

شماره دانشجویی: ۹۶۱۳۶۶۶۱۱۹

استاد: دکتر مهدی سعادتمند طرزجان

تاریخ تحویل: ۹۷/۴/۲۲

توضیحات:

در الگوریتم Chan - Vese، یک تصویر به نام مدل داشتیم که از نظر ابعاد دقیقا برابر با تصویر اصلی بود و به دو ناحیه داخلی و خارجی تقسیم می شد که مقادیر ناحیه داخلی و خارجی آن، برابر با میانگین وزن دار پیکسل های داخل منحنی و میانگین وزن دار پیکسل های خارج منحنی بودند. در این حالت از اطلاعات پیکسل های همسایه استفاده ای نمیشد و الگوریتم به صورت global انجام می پذیرفت. در الگوریتم LBF می خواهیم اطلاعات local را مورد استفاده قرار دهیم و مدل را به کمک یک patch طراحی نماییم. در نهایت معادله اویلر - لاگرانژ آن را بدست آورده و مرز اولیه به سمت ناحیه داخل تصویر همگرا شده و عمل تقطیع انجام می پذیرد.

قسمت الف: پیاده سازی

مرحله اول:

ابتدا تصویر رنگی را به تصویر خاکستری تبدیل می کنیم. سپس مقادیر سطح خاکستری پیکسل های آن را بین 0 و 1 تنظیم می کنیم.

```
if size(Img,3) == 3
    Img = rgb2gray(Img);
end
I = double(Img);
```

مرحله دوم:

یک کرنل گوسی طراحی می کنیم. برای طراحی این کرنل از رابطه زیر استفاده می شود:

$$K_\sigma(\mathrm{x}) = \frac{1}{(2\pi)^{n/2}\sigma^n} e^{-|\mathbf{x}|^2/2\sigma^2}$$

کد آن به صورت زیر می باشد:

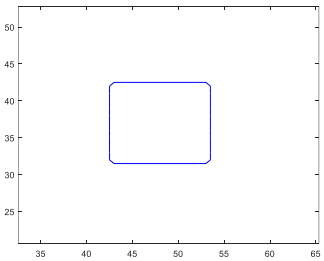
```
K = fspecial('gaussian',  1 + 4 * Kernel_Sigma, Kernel_Sigma);
```

مرحله سوم:

یک مرز اولیه (φ_0) طراحی می کنیم. مدل اولیه می تواند دایره، مربع یا هر شکل دیگری باشد. در این پروژه از مدل اولیه مستطیلی استفاده شده است. کد آن به صورت زیر می باشد:

```
phi = ones(r, c) .* -2;
phi(25:55,55:65) = 2;
```

خروجی قطعه کد بالا به صورت زیر می باشد:



مرحله چهارم:

در این قسمت مقدار تابع $H_\varepsilon(\varphi)$ را طبق رابطه زیر محاسبه می کنیم. این تابع به دلیل نرم بودن در لبه ها، خاصیت مشتق پذیری دارد.

$$H_\varepsilon(\varphi) = \frac{1}{2} \times (1 + \frac{2}{\pi} \arctan(\frac{\varphi}{\varepsilon}))$$

کد آن به صورت زیر می باشد:

```
H_eps = (1 + (2/pi) * atan(phi./eps)) / 2;
```

مرحله پنجم:

مقدار تابع $\delta_\varepsilon(\varphi)$ را طبق رابطه زیر محاسبه می کنیم:

$$\delta_\varepsilon(\varphi) = \frac{1}{\pi} \times (\frac{\varepsilon}{\varepsilon^2 + \varphi^2})$$

کد آن به صورت زیر می باشد:

```
Delta_eps = (1 / pi) .* (Eps ./ (Eps^2 + phi.^2));
```

مرحله ششم:

در این قسمت مقدار میانگین وزن دار داخل و خارج منحنی را با توجه به کرنل یا patch محاسبه می کنیم.

$$f_1(\mathrm{x}) = \frac{K_\sigma(\mathrm{x}) * [H_\varepsilon(\phi(\mathrm{x}))I(\mathrm{x})]}{K_\sigma(\mathrm{x}) * H_\varepsilon(\phi(\mathrm{x}))}$$

$$f_2(\mathrm{x}) = \frac{K_\sigma(\mathrm{x}) * [(1 - H_\varepsilon(\phi(\mathrm{x})))I(\mathrm{x})]}{K_\sigma(\mathrm{x}) * [1 - H_\varepsilon(\phi(\mathrm{x}))]}$$

کد آن به صورت زیر می باشد:

```
F1 = conv2(I .* H_eps , K , 'same') ./ conv2(H_eps , K , 'same');
F2 = conv2(I .* (1 - H_eps) , K , 'same') ./ conv2((1 - H_eps) , K , 'same');
```

مرحله هفتم:

در این قسمت معادله اویلر لاگرانژ را محاسبه می کنیم که شامل دو ترم Region و Regulator می باشد. و سپس نیروی کل را بدست می آوریم.
معادله اویلر لاگرانژ برای ترم Region الگوریتم LBF به صورت زیر می باشد:

$$\frac{\partial \varphi}{\partial t} = -\delta_\varepsilon(\varphi)(I(x)^2(\lambda_1 - \lambda_2) - 2I(x)\big(K * (\lambda_1 f_1 - \lambda_2 f_2)\big) + \Big(K * \big(\lambda_1 {f_1}^2 - \lambda_2 {f_2}^2\big)\Big)$$

معادله اویلر لاگرانژ برای ترم Regulator الگوریتم LBF به صورت زیر می باشد:

$$\frac{\partial \varphi}{\partial t} = \nu\delta_\varepsilon(\varphi)\nabla \cdot \frac{\nabla\varphi}{|\nabla\varphi|} + \mu(\nabla^2\varphi\ \nabla \cdot \frac{\nabla\varphi}{|\nabla\varphi|})$$

کد آن به صورت زیر می باشد:

```
T_Region = -Delta_eps .* (I.^2 .* (L1 - L2)
           - 2 * I .* conv2((L1 * F1 - L2 * F2), K, 'same')
           + conv2((L1 * F1 .^ 2 - L2 * F2 .^ 2), K, 'same'));

T_Regulator = nu .* Delta_eps .* kappa(phi) + mu .* (del2(phi) - kappa(phi));
```

تابع $\kappa(\varphi)$ به صورت زیر می باشد:

```
function k = kappa(I)

    I = double(I);

    [Model_G_X ,Model_G_Y]  = gradient(I);
    Norm = sqrt(Model_G_X.^2 + Model_G_Y.^2 + eps);
    Model_NG_X = Model_G_X ./ Norm;
    Model_NG_Y = Model_G_Y ./ Norm;

    [Model_G_XX ,~] = gradient(Model_NG_X);
    [~ , Model_G_YY] = gradient(Model_NG_Y);

    Divergence  = Model_G_XX + Model_G_YY;
    k = Divergence;
end
```

مرحله هشتم:

در این قسمت معادله تکامل منحنی را نوشته و آن را آپدیت می کنیم.

$$\varphi^{i+1} = \varphi^i + \eta\frac{\partial \varphi}{\partial t}$$

کد آن به صورت زیر می باشد:

```
phi = phi + eta .* (T_Region + T_Regulator);
```

قسمت ب: نتایج:

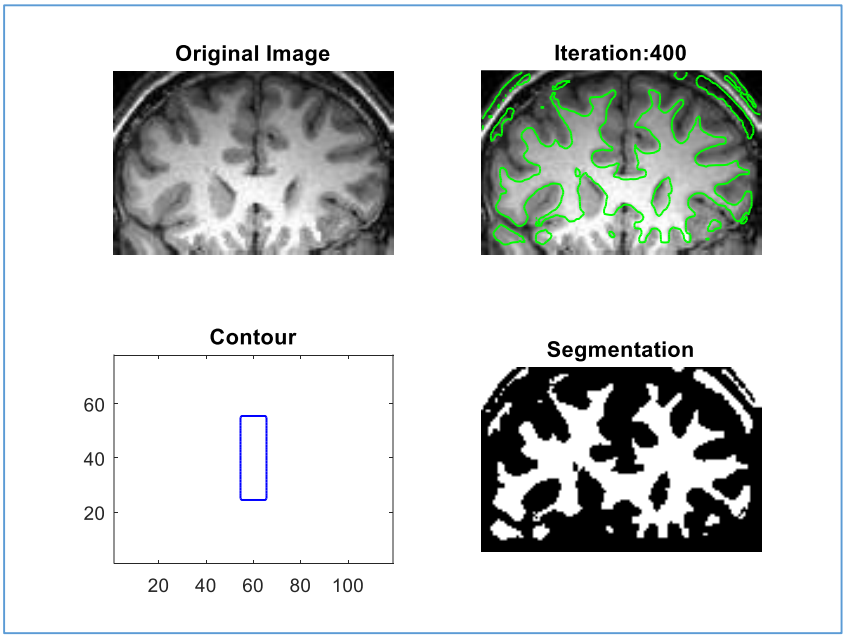
در این قسمت ۶ تصویر آزمایشی داریم که برای هریک از آن ها، هایپر پارامتر ها را به صورت منحصر بفرد تنظیم کرده ایم.

تصویر اول:

پارامتر ها:

```
Eps = 1;
eta = 0.1;
Kernel_Sigma = 3;
Iteration=400;
phi(25:55,55:65) = 2;
L1 = 1;
L2 = 2;
Mu = 1;
Nu = 0.003 * 255 ^ 2;
```

نتیجه الگوریتم LBF برای تصاویر آزمایشی ۱ به صورت زیر می باشد:



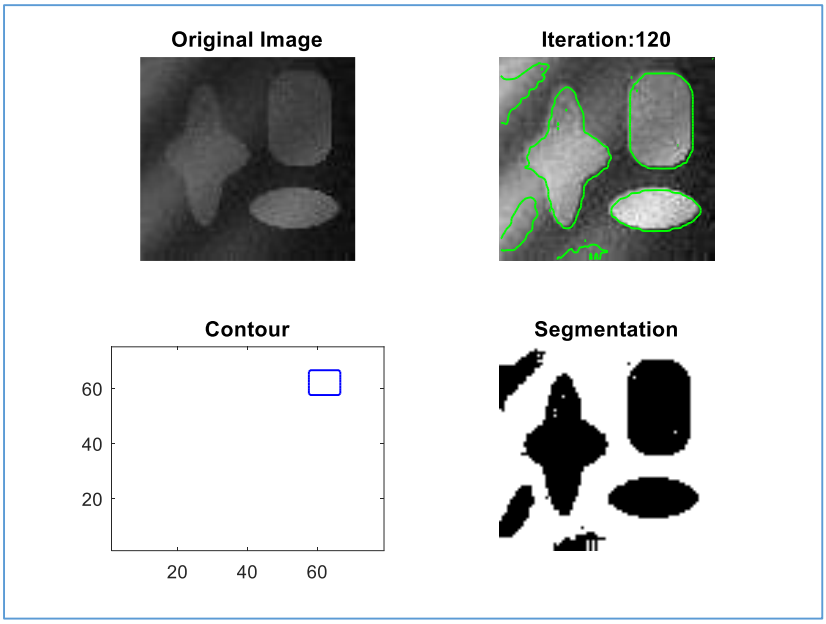
تصویر دوم:

پارامتر ها:

```
Eps = 1;
eta = 0.1;
Kernel_Sigma = 3;
```

```
Iteration = 120;
phi(58:66,58:66) = 2;
L1 = 1;
L2 = 1;
Mu = 1;
Nu = 0.001 * 255 ^ 2;
```

نتیجه الگوریتم LBF برای تصاویر آزمایشی ۲ به صورت زیر می باشد:



تصویر سوم:

پارامتر ها:

```
Eps = 1;
eta = 0.1;
Kernel_Sigma = 3;
Iteration = 250;
phi(55:60,75:79) = 2;
L1 = 1;
L2 = 1;
Mu = 1;
Nu = 0.003 * 255 ^ 2;
```

نتیجه الگوریتم LBF برای تصاویر آزمایشی۳ به صورت زیر می باشد:

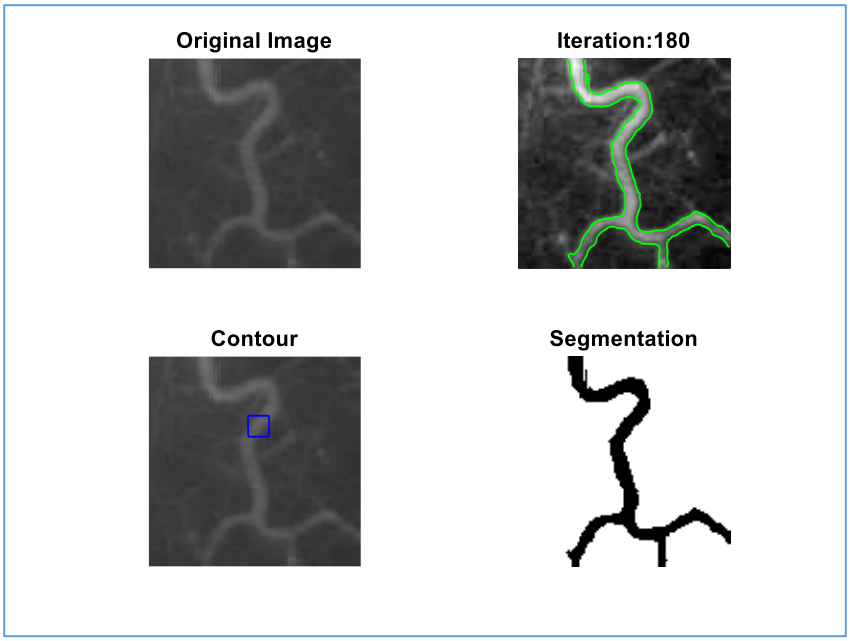
تصویر چهارم:

پارامتر ها:

```
Eps = 1;
eta = 0.1;
Kernel_Sigma = 3;
```

```
Iteration = 180;
phi(55:60,75:79) = 2;
L1 = 1;
L2 = 1;
Mu = 1;
Nu = 0.001 * 255 ^ 2;
```

نتیجه الگوریتم LBF برای تصاویر آزمایشی۴ به صورت زیر می باشد:

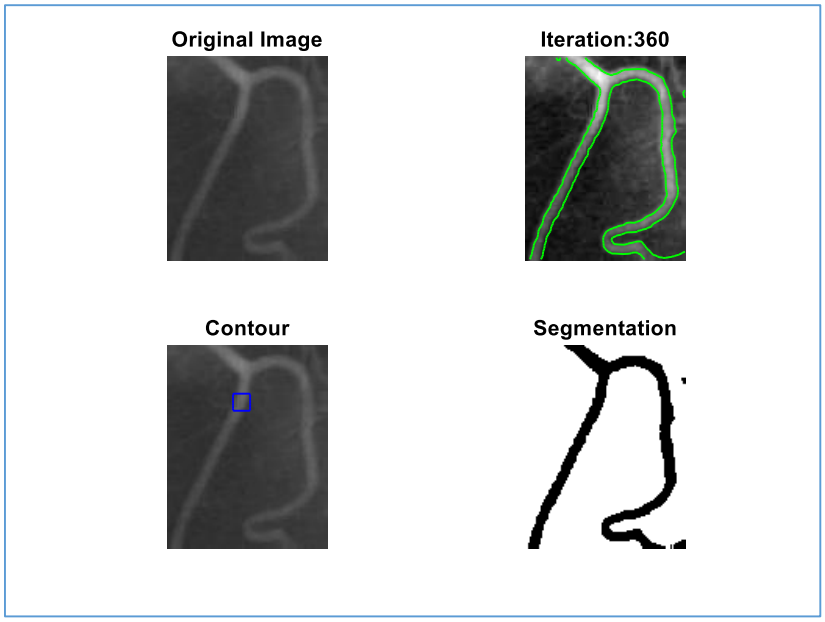


تصویر پنجم:

پارامتر ها:

```
Eps = 1;
eta = 0.1;
Kernel_Sigma = 3;
Iteration = 370;
phi(55:60,75:79) = 2;
L1 = 1;
L2 = 1;
Mu = 1;
Nu = 0.001 * 255 ^ 2;
```

نتیجه الگوریتم LBF برای تصاویر آزمایشی۵ به صورت زیر می باشد:

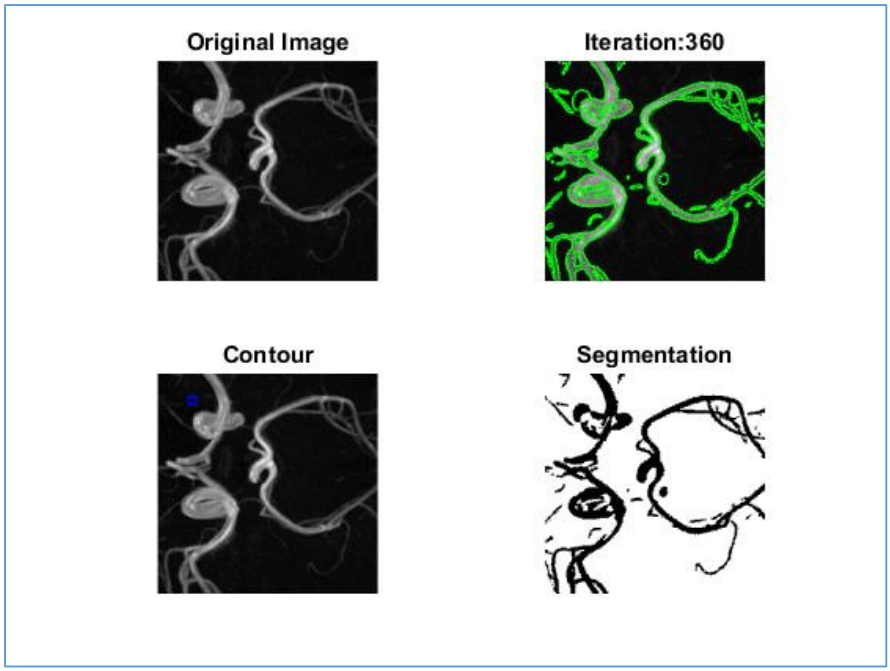


تصویر ششم:

پارامتر ها:

```
Eps = 1;
eta = 0.1;
Kernel_Sigma = 3;
Iteration = 370;
phi(55:60,75:79) = 2;
L1 = 1;
L2 = 1;
Mu = 1;
Nu = 0.001 * 255 ^ 2;
```

نتیجه الگوریتم LBF برای تصاویر آزمایشی ۶ به صورت زیر می باشد:



پیوست:

فایل اصلی main.m

```
clear all;
close all;
clc;

ID = 6;

Img = imread(strcat('images/', int2str(ID),'.jpg'));

if size(Img,3) == 3
    Img = rgb2gray(Img);
end

I = double(Img);
[r, c] = size(I);

phi = ones(r, c) .* -2;

%% Hyper Parameters
Eps = 1;
eta = 0.1;
Kernel_Sigma = 3;

if ID == 1
    Iteration=400;       phi(25:55,55:65)=2;       L1=1;       L2=2;
    mu=1;               nu=0.003*255^2;

elseif ID==2
    Iteration=120;       phi(58:66,58:66)=2;       L1=1;       L2=1;
    mu=1;               nu=0.001*255^2;

elseif ID==3
    Iteration=1000;       phi(30:100,30:100)=2;       L1=1;       L2=1;
    mu=1;               nu=0.001*255^2;

elseif ID==4
    Iteration=180;       phi(32:42,53:63)=2;       L1=1;       L2=1;
    mu=1;               nu=0.001*255^2;

elseif ID==5
    Iteration=370;       phi(32:42,43:53)=2;       L1=1;       L2=1;
    mu=1;               nu=0.001*255^2;
```

```
elseif ID==6
    Iteration=370;       phi(32:42,43:53)=2;        L1=1;       L2=1;
    mu=1;               nu=0.001*255^2;
end
%%

K = fspecial('gaussian',  1 + 4 * Kernel_Sigma, Kernel_Sigma);

figure;
subplot(2,2,1);
imshow(Img);
title('Original Image');

subplot(2,2,3);
imshow(Img);
hold on;
contour(phi, [0 0], 'y','LineWidth',1);
hold off;
title('Contour');

subplot(2,2,2);
imshow(I,[],'initialmagnification','fit');
hold on;

for i = 1 : Iteration

    H_eps = (1 + (2/pi) * atan(phi ./ Eps)) / 2;
    Delta_eps = (1 / pi) .* (Eps ./ (Eps^2 + phi.^2));

    F1 = conv2(I .* H_eps , K , 'same') ./ conv2(H_eps , K , 'same');
    F2 = conv2(I .* (1 - H_eps) , K , 'same') ./ conv2((1 - H_eps) , K , 'same');

    T_Region = -Delta_eps .* (I.^2 .* (L1 - L2) - 2 * I .* conv2((L1 * F1 - L2 * F2), K,
'same') + conv2((L1 * F1 .^ 2 - L2 * F2 .^ 2), K, 'same'));
    T_Regulator = nu .* Delta_eps .* kappa(phi) + mu .* (del2(phi) - kappa(phi));

    phi = phi + eta .* (T_Region + T_Regulator);

    if(mod(i,20)==0)
        imshow(Img,[],'initialmagnification','fit');
        hold on;
        contour(phi,[0 0],'g','LineWidth',1);
        title(strcat('Iteration: ', num2str(i)));
        drawnow;
    end
end
```

```
result = phi <= 0;

subplot(2,2,4);
imshow(result);
title('Segmentation');
```

تابع kappa.m

```
function k = kappa(I)

    I = double(I);

    [Model_G_X ,Model_G_Y]  = gradient(I);
    Norm = sqrt(Model_G_X.^2 + Model_G_Y.^2 + eps);
    Model_NG_X = Model_G_X ./ Norm;
    Model_NG_Y = Model_G_Y ./ Norm;

    [Model_G_XX ,~] = gradient(Model_NG_X);
    [~ , Model_G_YY] = gradient(Model_NG_Y);

    Divergence  = Model_G_XX + Model_G_YY;
    k = Divergence;
end
```