Math 6590 Project IV: Some analysis about the snake model and implementation of the geodesic active contour model

Name: Li Dong
RIN: 661243168

Apr 24, 2016

Abstract

This report discusses two classical model in image segmentation - snake active contour (SAC) model and geodesic active contour (GAC) model. We first derived the gradient flow, or the first variation principle, for the energy function used in the SAC model. Then the GAC model is analysed and implemented with finite difference method. Details of the finite difference schemes are also displayed. At last, a image segmentation demonstration of the GAC model is presented. Some details about tuning of the GAC model, such as initial level set curve and reinitialize the level set curve during iterations, are discussed.

1 Problems

1. Derive the first variational principle for the following energy used in the snake active contour (SAC) model [1].
$$J(C) = \int_a^b |\frac{\partial C}{\partial q}(t,q)|^2 \mathrm{d}q + \lambda \int_a^b g^2(|\nabla I(C(t,q))|)\mathrm{d}q \tag{1}$$
and show it has the following gradient flow
$$\frac{\partial C}{\partial t} = \left[\kappa \left|\frac{\partial C}{\partial q}\right|^2 - \lambda\langle g\nabla g, \vec{N}\rangle\right]\vec{N} - \left[\lambda\langle g\nabla g, \vec{T}\rangle - \langle \vec{T}, \frac{\partial^2 C}{\partial q^2}\rangle\right]\vec{T} \tag{2}$$

2. Describe the geodesic active contour (GAC) model [2] and the numerical method (including details of the finite difference schemes) for image segmentation. Implement the geodesic active contour model and test the attached image (use the clean image and the noisy image).

2 Methodology

2.1 Gradient flow for SAC model

In order to derive the first variational principle for Eq.(1), we need to take the Gateaux derivative of Eq.(1) with respect to t as follows.

First, we denote $\frac{\partial C}{\partial q}(t,q)$ as C_q and $g(|\nabla I(C(t,q))|)$ simply as g. Also, we need to use some basic relations from curve theory, such as $\vec{T_q} = \vec{T_s} * |C_q|$, $\vec{T_s} = \kappa\vec{N}$ and $\vec{T} = \frac{C_q}{|C_q|}$, where s denotes the arc-length

of $C(t)$ and $\vec{T}$ and $\vec{N}$ denote unit tangent and unit normal of $C(t)$ at corresponding locations.

$$\begin{aligned}
\frac{\partial J(C)}{\partial t}\Big|_{t=0} &= \frac{\partial}{\partial t}\int_a^b |C_q|^2 \mathrm{d}q + \lambda \frac{\partial}{\partial t}\int_a^b g^2 \mathrm{d}q\Big|_{t=0}, &(3)\\
&= 2\int_a^b \langle |C_q|\vec{T}, C_{qt}\rangle \mathrm{d}q + 2\lambda \int_a^b \langle g\nabla g, C_t\rangle \mathrm{d}q\Big|_{t=0}, &(4)\\
&= -2\int_a^b \langle \langle \vec{T}, C_{qq}\rangle \vec{T} + |C_q|\vec{T}_q, C_t\rangle \mathrm{d}q + 2\lambda \int_a^b \langle g\nabla g, C_t\rangle \mathrm{d}q\Big|_{t=0}, &(5)\\
&= -2\int_a^b \langle \langle \vec{T}, C_{qq}\rangle \vec{T}, C_t\rangle \mathrm{d}q - 2\int_a^b \kappa |C_q|^2 \langle \vec{N}, C_t\rangle \mathrm{d}q + 2\lambda \int_a^b \langle g\nabla g, C_t\rangle \mathrm{d}q\Big|_{t=0}, &(6)\\
&= -2\int_a^b \langle \langle \vec{T}, C_{qq}\rangle \vec{T} + \kappa |C_q|^2 \vec{N} - \lambda g\nabla g, C_t\rangle \mathrm{d}q\Big|_{t=0}, &(7)\\
&= -2\int_a^b \langle \langle \vec{T}, C_{qq}\rangle \vec{T} + \kappa |C_q|^2 \vec{N} - \lambda \langle g\nabla g, \vec{N}\rangle \vec{N} - \lambda \langle g\nabla g, \vec{T}\rangle \vec{T}, \overset{0}{C}_t\rangle \mathrm{d}q. &(8)
\end{aligned}$$

where the boundary terms vanish after integrating by parts because the curve C is closed. Now it is clear that the gradient flow can be written as,

$$C_t = [\kappa |C_q|^2 - \langle g\nabla g, \vec{N}\rangle]\vec{N} - [\langle g\nabla g, \vec{T}\rangle - \langle \vec{T}, C_{qq}\rangle]\vec{T}. \tag{9}$$

This completes the first problem for this project.

2.2 Gradient flow and finite difference strategy for GAC model

Since the SAC model is not invariant with respect to parameterization, the GAC model was developed and popularized. Specially, level set method is commonly used to implement the GAC model for convenience. The GAC model can be stated as, given an initial curve $\overset{0}{C}$,

$$\min_C \int_0^q g(|\nabla I|)|C_q|\mathrm{d}q, \tag{10}$$

where g serves as an edge detector. After deriving the first variation principle of Eq. (10), we can have the following gradient flow,

$$\begin{aligned}
\frac{\partial C}{\partial t} &= (g(C)(\kappa+\alpha) - \nabla g(C)\vec{N})\vec{N}, &(11)\\
C(t,q)\Big|_{t=0} &= \overset{0}{C}(q), &(12)
\end{aligned}$$

where α should be big enough such that $\kappa + \alpha > 0$.

Next, we represent the above gradient flow with level set method. If we denote the level set function $u(t, C(t,q)) = 0$, we have

$$\begin{aligned}
\frac{\partial C}{\partial t} &= F\vec{T}, &(13)\\
C(t,q)\Big|_{t=0} &= \overset{0}{C}(q), &(14)
\end{aligned}$$

where $F = g(|\nabla I|)(div(\frac{\nabla u}{|\nabla u|}) + \alpha) + (\nabla g(|\nabla I|)\frac{\nabla u}{|\nabla u|})$ and g is the edge detector, $g(|\nabla I|) = \frac{1}{1+|\nabla I|^2}$. Thus, we can write the gradient flow in terms of the level set method as follows.

$$\begin{aligned}
\frac{\partial u}{\partial t} &= g(|\nabla I|)|\nabla u|\kappa + \alpha |\nabla u| g(|\nabla I|), &(15)\\
u(0,x,y) &= \overset{0}{u}(x,y), &(16)
\end{aligned}$$

where $\kappa = div(\frac{\nabla u}{|\nabla u|})$ and g is the edge detector, $g(|\nabla I|) = \frac{1}{1+|\nabla I_\epsilon|^2}$ and I_ϵ is a smoothed I, specifically a Gaussian kernel for this report. Also, the level set function u is a signed distance function in this case and takes negative values inside the level set, positive values outside the level set and zeros on the level set. The algorithm is as follows.

1. Fixed an inital $\overset{0}{u}$ as a signed distance function.
2. Apply Eq.(15).
3. Extract the zero level set of u.
4. We run the above two steps a few times and reinitialize the level set function with the following rule.

$$\frac{\partial \phi}{\partial t} + sign(\phi)(|\nabla \phi| - 1) = 0, \tag{17}$$

$$\phi(0, x, y) = u(x, y). \tag{18}$$

5. We run the previous three steps until $\frac{||u^{n+1} - u^n||}{||u^n||} < \epsilon$ or the specified maximum iteration number is reached.

In the end, we describe the finite difference strategy. It is noted that Eq.(15) consists of a parabolic part and a hyperbolic part on the right hand side. For the parabolic part, regular finite difference strategy is enough; For the hyperbolic part, upwind difference is used. First let's define some notations. We use the following to represent the forward, backward and central difference for tackling with the parabolic part.

$$\Delta_x^+ u_{ij} = u_{i+1,j} - u_{ij}, \tag{19}$$

$$\Delta_x^- u_{ij} = u_{i,j} - u_{i-1,j}, \tag{20}$$

$$\Delta_x^c u_{ij} = \frac{u_{i+1,j} - u_{i-1,j}}{2}, \tag{21}$$

$$\Delta_y^+ u_{ij} = u_{i+1,j} - u_{ij}, \tag{22}$$

$$\Delta_y^- u_{ij} = u_{i,j} - u_{i-1,j}, \tag{23}$$

$$\Delta_y^c u_{ij} = \frac{u_{i+1,j} - u_{i-1,j}}{2}. \tag{24}$$

Moreover, we use the following notations to represent the upwind difference for tackling with the hyperbolic part.

$$\nabla^+ u_{ij} = \sqrt{max(\Delta_x^- u_{ij}^n, 0)^2 + min(\Delta_x^+ u_{ij}^n, 0)^2 + max(\Delta_y^- u_{ij}^n, 0)^2 + min(\Delta_y^+ u_{ij}^n, 0)^2}, \tag{25}$$

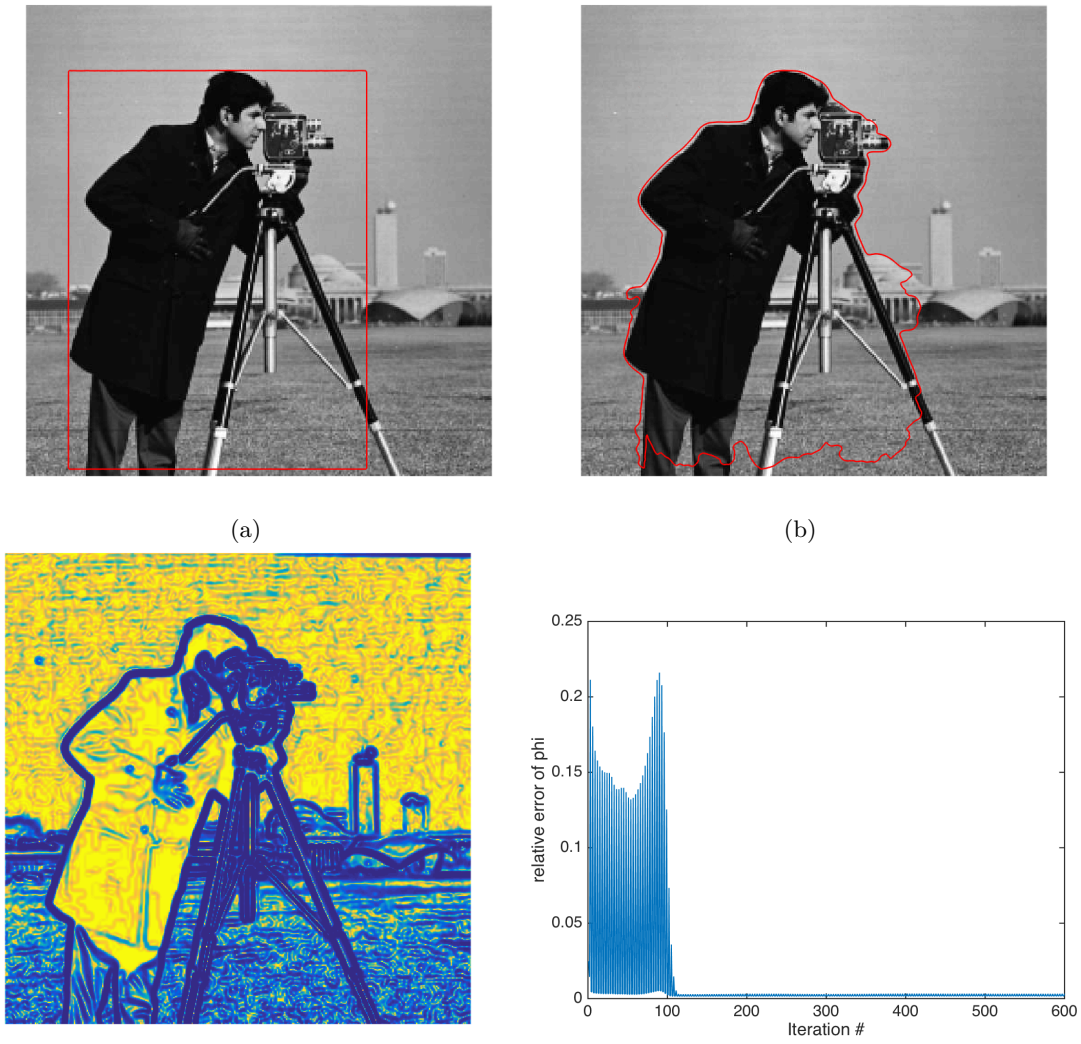
$$\nabla^- u_{ij} = \sqrt{max(\Delta_x^+ u_{ij}^n, 0)^2 + min(\Delta_x^- u_{ij}^n, 0)^2 + max(\Delta_y^+ u_{ij}^n, 0)^2 + min(\Delta_y^- u_{ij}^n, 0)^2}. \tag{26}$$

In addition, we have $\kappa = \Delta^- \frac{\Delta^+ u}{\sqrt{|\Delta^+ u|^2 + \epsilon}}$. Now that we have all the ingredients for Eq.(15), we can write its discretized form.

$$\begin{aligned} u_{ij}^{n+1} =& u_{ij}^n + dt(g_{ij}\kappa_{ij}^n \sqrt{(\Delta_x^c u_{ij}^n)^2 + (\Delta_y^c u_{ij}^n)^2} + \alpha(max(g_{ij}, 0)\nabla^+ + min(g_{ij}, 0)\nabla^-)u_{ij}^n + \\ & max(g_x(i,j), 0)\nabla_x^- u_{ij}^n + min(g_x(i,j), 0)\nabla_x^+ u_{ij}^n + max(g_y(i,j), 0)\nabla_y^- u_{ij}^n + min(g_y(i,j), 0)\nabla_y^+ u_{ij}^n). \end{aligned} \tag{27}$$

3 Results

We ran the algorithm for the cameraman image without and with noise. It is found that the reinitialization has to be called very frequently, every three iterations in this case; Otherwise, the algorithm tends to blow up at some point. Figures 1a and 2a display the initial zero level set as a red rectangular.



(a) (b)

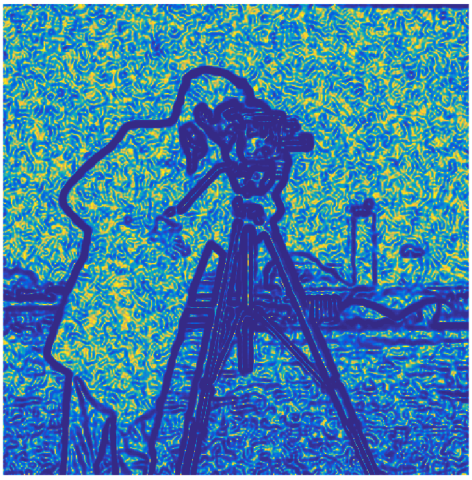
(c) (d)

Figure 1: Case with no noise. (a) Red rectangle denotes the initial level set, (b) Result at the end of iterations, (c) plot of the edge detector g, (d) convergence plot where the vertical axis displays $\frac{||u^{n+1}-u^n||}{||u^n||}$.

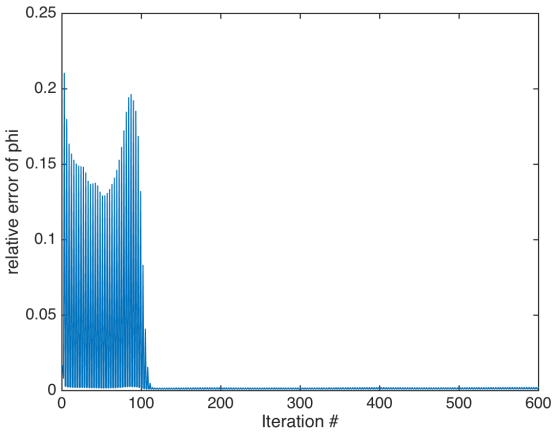
Figures 1b and 2b display the results after convergence. We can see that the level set curve in the case with noise is not as good as the one without noise. This is because that there are more local minima for the algorithm to "fight with" in the noisy case. This can be seen clearer from Figures 1c and 2c. Although the edge of the cameraman is quite strong in both cases, the noisy case has more edge-like noise in the background. Interestingly, when watching the video clips GAC_0.avi and GAC_1.avi, the level set curve was caught by the tall building on the right of the cameraman for a while and then the curve "struggled" to the thicker edges around the cameraman. Same convergence criteria is used for both cases - $\frac{||u^{n+1}-u^n||}{||u^n||} < 0.00015$ and maximum iteration $= 600$, whichever is satisfied first. In both cases, the convergence is relatively slow, as Figures 1d and 2d show. No apparent drop after around 100 iterations, though the level set curve is still moving toward the target slowly. Both cases were terminated after 600 iterations. Also, every time the level set curve is initialized, the convergence curve bumps up which makes the convergence curve appears oscillating.

(a)

(b)



(c)



(d)

Figure 2: Case with noise. (a) Red rectangle denotes the initial level set, (b) Result at the end of iterations, (c) plot of the edge detector g, (d) convergence plot.

4 Observation and Conclusions

1. From the gradient flow expression, it is clear that SAC model is not invariant to parameterization. Thus GAC is a better model in this sense.
2. The initial level set curve for the GAC model is very important. If it is very near the segmented edges, the algorithm can converge much faster.
3. The GAC model is very easy to be trapped to local minima and this is especially obvious when there is noise.
4. Reinitialization of the level set curve is very essential for the stability of the GAC algorithm. The more frequent the curve is initialized is more stable the algorithm can be, yet the efficiency is sacrificed.

References

[1] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[2] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *International journal of computer vision*, 22(1):61–79, 1997.

Appendix

MATLAB codes.

main.m calls other functions to compute the edge detector and gradient flow and save results.

```
clear
close all

[I_0, I_n]=LoadData;

I=I_0;

% initial level set

% rectangular
phi=zeros(size(I));
set0_row_top=71; set0_col_lef=47;
set0_row_bot=500; set0_col_rig=375;
phi(set0_row_top:set0_row_bot,set0_col_lef:set0_col_rig)=2;
phi(set0_row_top,set0_col_lef:set0_col_rig)=1;
phi(set0_row_bot,set0_col_lef:set0_col_rig)=1;
phi(set0_row_top:set0_row_bot,set0_col_lef)=1;
phi(set0_row_top:set0_row_bot,set0_col_rig)=1;
phi=1-phi;

dt=0.015;
alpha=10;
eps=1e-2;
iter=600;
h=figure;
```

```
conv=figure;
[g,g_max,g_min,grad_g_max,grad_g_min]=g_edge(I);
figure; imagesc(g); axis off
pbaspect([size(I),1]);
saveas(gcf,'edges_0.png');

conv_phi=zeros(iter,1);
mov_interval=10; % save 1 snapshot from each 10 iterations
mov(1:1 + iter/mov_interval)=struct('cdata',[],'colormap',[]);
mov_cnt=1;
for i=1:iter
    phi_prev=phi;
    phi=GAC(alpha,dt,eps,phi,g,g_max,g_min,grad_g_max,grad_g_min);

    figure(h);
    imagesc(I); colormap('gray'); axis off
    pbaspect([size(I),1]);
    hold on
    contour(phi,[0,0],'linecolor','red');
    drawnow

    if (mod(i,3)==0)
       phi=Reinitial2D(phi,10);
       figure(h);
       imagesc(I); colormap('gray'); axis off
       pbaspect([size(I),1]);
        hold on
        contour(phi,[0,0],'linecolor','red');
        drawnow
    end

    if(mod(i,mov_interval)==0 || i==1)
        mov(mov_cnt)=getframe(h);
        mov_cnt=mov_cnt+1;
    end

    conv_phi(i)=norm(phi_prev(:)-phi(:))/norm(phi_prev);
    conv_phi(i)
    if(conv_phi(i)<=dt/100)
        display('Converged!');
        break
    elseif(isnan(conv_phi(i)))
        display('Diverged!');
        break
    end
    figure(conv);
    plot(conv_phi); xlabel('Iteration #'); ylabel('relative error of phi');
    set(gca,'fontsize',15)

    if (i==1)
        saveas(h,sprintf('contour_n_%d.png',i));
    end
    if (i==iter)
        display('Done!');
```

```
        saveas(h,sprintf('contour_n_%d.png',i));
        saveas(conv,'conv_n.png');
        movie2avi(mov,'GAC_n.avi','compression','None');
    end
end
```

g_edge.m computes all the matrices related to the edge detector.

```
function [g,g_max,g_min,grad_g_max,grad_g_min]=g_edge(I)

Iblur = imgaussfilt(I, 2);
grad_I=grad(Iblur,[0 0]);
g=1./(1+grad_I(:,:,1).^2+grad_I(:,:,2).^2);
grad_g=grad(g,[0 0]);

g_max=g; g_max(g<0)=0;
g_min=g; g_min(g>0)=0;
grad_g_max=grad_g; grad_g_max(grad_g<0)=0;
grad_g_min=grad_g; grad_g_min(grad_g>0)=0;
```

GAC.m computes the main step of the gradient flow.

```
function [phi]=GAC(alpha,dt,eps,phi,g,g_max,g_min,grad_g_max,grad_g_min)

grad_phi_cc=grad(phi,[0 0]); % central
grad_phi_bb=grad(phi,[2 2]); % backward x, backward y
grad_phi_ff=grad(phi,[1 1]); % forward x, forward y
K=div(grad_phi_ff(:,:,1),grad_phi_ff(:,:,2))...
    ./sqrt(grad_phi_ff(:,:,1).^2+grad_phi_ff(:,:,2).^2+eps^2);

% upwind
grad_phi_bb_max=grad_phi_bb; grad_phi_bb_max(grad_phi_bb<0)=0;
grad_phi_ff_min=grad_phi_ff; grad_phi_ff_min(grad_phi_ff>0)=0;
upwind_pos_phi=sqrt(grad_phi_bb_max(:,:,1).^2+grad_phi_ff_min(:,:,1).^2+...
                grad_phi_bb_max(:,:,2).^2+grad_phi_ff_min(:,:,2).^2);
grad_phi_ff_max=grad_phi_ff; grad_phi_ff_max(grad_phi_ff<0)=0;
grad_phi_bb_min=grad_phi_bb; grad_phi_bb_min(grad_phi_bb>0)=0;
upwind_neg_phi=sqrt(grad_phi_ff_max(:,:,1).^2+grad_phi_bb_min(:,:,1).^2+...
                grad_phi_ff_max(:,:,2).^2+grad_phi_bb_min(:,:,2).^2);

phi=phi+dt*( g.*K.*sqrt((grad_phi_cc(:,:,1)).^2+(grad_phi_cc(:,:,2)).^2) ...
    + alpha*(g_max.*upwind_pos_phi+g_min.*upwind_neg_phi) + ...
    + grad_g_max(:,:,1).*grad_phi_bb(:,:,1) + grad_g_min(:,:,1).*grad_phi_ff(:,:,1) ...
    + grad_g_max(:,:,2).*grad_phi_bb(:,:,2) + grad_g_min(:,:,2).*grad_phi_ff(:,:,2) );
```

div.m is the divergence operator.

```
function fd = div(Px,Py)
% backward x, backward y
nbdims = 2;
if size(Px,1)==1 || size(Px,2)==1
    nbdims = 1;
end
```

```
fx = Px-Px([1 1:end-1],:);
if nbdims>=2
    fy = Py-Py(:,[1 1:end-1]);
end

if nbdims==2
    fd = fx+fy;
else
    fd = fx;
end
```

grad.m is the gradient operator.

```
function [fx] = grad(M,opt)
% 1:forward, 2:backward, 0:central

switch opt(1) % x dir
    case 1
        fx = M([2:end end],:)-M;
    case 2
        fx = M-M([1 1:end-1],:);
    case 0
        fx = ( M([2:end end],:) - M([1 1:end-1],:) )./2;
end

switch opt(2) % y dir
    case 1
        fy = M(:,[1 1:end-1])-M;
    case 2
        fy = M-M(:,[2:end end]);
    case 0
        fy = ( M(:,[2:end end])-M(:,[1 1:end-1]) )./2;
end

fx = cat(3,fx,fy);

end
```

No changes have been made to Reinitial2D.m, thus not displayed here.