**EECE 8395**

**Project 7**

**LevelSets**

**Tahsin Reasat**

**ID: 000614908**

Levelset-GVF on small test image:

The GVF field



The error decreases but gets stuck at a high value of 1.4 and the resultant contour is



LevelSet Applied on Pepper image:

There are small one pixel contours that won’t go away.







Code

FastMarch

function [dmapout,nbin,nbout]=FastMarch(img,maxdist,getnb,nbi)

global dmap Active dmapi heap

[r,c]=size(img);

d=1;

heap = HeapInit2(10000);

dmapi=img;

dmap = 3e8\*ones(r,c,d);

dmap(dmapi(:)==0)=0;

Active = ones(r,c);

if nargin<4 || isempty(nbi)

nbi.q = [1:r\*c; dmap(:)'];

nbi.len = length(nbi.q);

end

InsertBorderVoxelsIntoHeap(dmapi,1,nbi)

if getnb

nb.q = zeros(2,r\*c\*d);

nb.len=0;

end

[node,dist]=HeapPop2;

while ~isempty(node) && dist<maxdist

if getnb

nb.len = nb.len+1;

nb.q(:,nb.len) = [node;dist];

end

Active(node)=0;

ProcessNeighborsEikonal(node,dmapi,1)

[node,dist] = HeapPop2();

while (~isempty(node))&&Active(node)==0

[node,dist] = HeapPop2();

end

end

% That gives us our foreground distance map. Now we do it again for background.

dmapin = dmap;

dmap = 3e8\*ones(r,c,d);

dmap(dmapi(:)==0)=0;

Active = ones(r,c,d);

% background

InsertBorderVoxelsIntoHeap(dmapi,-1,nbi);

if getnb

nbin = nb;

nb.len=0;

end

[node,dist] = HeapPop2();

while ~isempty(node) && dist<maxdist

if getnb

nb.len = nb.len+1;

nb.q(:,nb.len) = [node;dist];

end

Active(node)=0;

ProcessNeighborsEikonal(node,dmapi,-1);

[node,dist] = HeapPop2();

while (~isempty(node))&&Active(node)==0

[node,dist] = HeapPop2();

end

end

%Then we combine the two results into our output distance map:

dmapout = dmap;

dmapout(dmapi(:)<0) = -dmapin(dmapi(:)<0);

if getnb

nbout = nb;

end

% mean(abs(dmapout(:)-img(:)));

% max(abs(dmapout(:)-img(:)));

function InsertBorderVoxelsIntoHeap(dmapi,mode,nbi)

global dmap Active Edges

if nargin<3 || isempty(nbi)

nbi=struct;

nbi.len=length(Active(:));

nbi.q=1:length(Active(:));

end

if mode==1

nodes =find(dmapi(:)<0); %foreground pixels

end

if mode==-1

nodes = find(dmapi(:)>0); %background pixels

end

if ~isempty(nbi)

nodes=intersect(nodes,nbi.q(1,1:nbi.len));

end

for i\_nodes=1:length(nodes)

node=nodes(i\_nodes);

node\_dist=dmapi(node);

neibs=Edges(node,:);

for i\_neibs=1:length(neibs)

if neibs(i\_neibs)

neib\_dist=dmapi(neibs(i\_neibs));

if neib\_dist\*node\_dist<0

%% get valid neighbours of the opposite class

if neibs(1) && sign(dmapi(neibs(1)))==mode

R=neibs(1);

else

R=0;

end

if neibs(2) && sign(dmapi(neibs(2)))==mode

L=neibs(2);

else

L=0;

end

if neibs(3) && sign(dmapi(neibs(3)))==mode

D=neibs(3);

else

D=0;

end

if neibs(4) && sign(dmapi(neibs(4)))==mode

U=neibs(4);

else

U=0;

end

%% calculate distance from neighbours

%% LR

if L==0 && R

x=abs(node\_dist)/(abs(dmapi(R))+abs(node\_dist));

end

if R==0 && L

x=abs(node\_dist)/(abs(dmapi(L))+abs(node\_dist));

end

if R==0 && L==0

x=Inf;

end

if L&&R

x=min([abs(node\_dist)/(abs(dmapi(L))+abs(node\_dist)) ...

abs(node\_dist)/(abs(dmapi(R))+abs(node\_dist))]);

end

%% DU

if D&&U

y=min([abs(node\_dist)/(abs(dmapi(D))+abs(node\_dist))...

abs(node\_dist)/(abs(dmapi(U))+abs(node\_dist))]);

end

if D==0 && U

y=abs(node\_dist)/(abs(dmapi(U))+abs(node\_dist));

end

if U==0 && D

y=abs(node\_dist)/(abs(dmapi(D))+abs(node\_dist));

end

if D==0 && U==0

y=Inf;

end

dist=sqrt((1/x^2+1/y^2)^-1);

dmap(node)=dist;

HeapInsert2(node,dist);

Active(node)=2;

end

end

end

end

function ProcessNeighborsEikonal(node,dmapi,mode)

global Edges Active dmap

neibs=Edges(node,:);

neibs=neibs(neibs~=0); % take nonzero neighbors

if mode==1

neibs=neibs(dmapi(neibs)<0); %foreground

end

if mode==-1

neibs=neibs(dmapi(neibs)>0); %background

end

for i=1:length(neibs)

if Active(neibs(i))==1

dist=dist\_calc(neibs(i));

if dmap(neibs(i))>dist

dmap(neibs(i))=dist;

HeapInsert2(neibs(i),dist)

end

end

end

function dist=dist\_calc(node)

global dmap Edges

neibs=Edges(node,:);

if neibs(3)==0

U\_ud=dmap(neibs(4));

end

if neibs(4)==0

U\_ud=dmap(neibs(3));

end

if neibs(3) && neibs(4)

U\_ud=min(dmap(neibs(3)),dmap(neibs(4)));

end

if neibs(1)==0

U\_lr=dmap(neibs(2));

end

if neibs(2)==0

U\_lr=dmap(neibs(1));

end

if neibs(1) && neibs(2)

U\_lr=min(dmap(neibs(1)),dmap(neibs(2)));

end

Us=[U\_ud,U\_lr];

Us=sort(Us);

dist=Us(1)+1;

if dist > Us(2)

dist = (Us(1) + Us(2) + sqrt(2-(Us(1)-Us(2))^2))/2;

End

LevelSet

function res = LevelSetGVF(img,res, sigma, errthrsh, maxiter, mu, gamma)

global Edges

mindist=2.1;

[r,c]=size(img);

d=1;

img = .5 - img;

g = fspecial('gaussian',[5,5],sigma);

imgblur = conv2(img,g,'same');

[Y,X] = meshgrid(1:c,1:r);

Y = Y(:);

X = X(:);

Edges =[Y<c,Y>1,X<r,X>1].\*(repmat([1:r\*c]',[1,4]) +repmat([r,-r,1,-1],[r\*c,1]));

grad = Gradient(imgblur,1:r\*c\*d);

ngrad = reshape(sum(grad.\*grad),[r,c]);

speed = exp(-ngrad/(.08));

figure(1); clf; colormap(gray(256));

image(speed\*1000);

hold on;

title('speed');

gradspeed = Gradient(speed,1:r\*c);

quiver(reshape(gradspeed(1,:),[r,c]),reshape(gradspeed(2,:),[r,c]),'b')

gradspeed = GVF(gradspeed,mu,[r,c]);

hold on

quiver(reshape(gradspeed(1,:),[r,c]),reshape(gradspeed(2,:),[r,c]),'g')

iter = 0;

nb = [];

while iter<maxiter

iter = iter+1;

figure(2);clf; colormap(gray(256))

hold off

image(speed\*1000);

hold on;

contour(res,[0,0],'r');

title('speed');

drawnow;

[res,nbin,nbout] = FastMarch(res,mindist,1,nb);

if iter>1

err = sum(abs(-res(nbinold.q(1,1:nbinold.len))-nbinold.q(2,1:nbinold.len)))+...

sum(abs(res(nboutold.q(1,1:nboutold.len))-nboutold.q(2,1:nboutold.len)))

if err<errthrsh

break;

end

end

nboutold = nbout;

nbinold = nbin;

figure(3);clf; colormap(gray(256))

hold off;

image(res\*10+127);

hold on;

contour(res,[0,0],'r');

title(['distance map iter=',num2str(iter)])

drawnow;

nb.q = [nbin.q(:,1:nbin.len),nbout.q(:,1:nbout.len)];

nb.len = size(nb.q,2);

nbspeed.q = nb.q(:,nb.q(2,1:nb.len)<=1);

nbspeed.len = size(nbspeed.q,2);

[kappa,ngrad,grad] = Curvature2(res,nbspeed);

node = nbspeed.q(1,1:nbspeed.len);

speedc=-speed(node).\*(max(ngrad,0.001)).\*(kappa+gamma) + sum(grad.\*gradspeed(:,node));

dt = 0.5/max(abs(speedc(:)));

res(node) = res(node) + dt\*speedc;

figure(4); clf; colormap(gray(256))

curvature = zeros(size(res));

curvature(node)=kappa;

image(curvature\*500+127);

title('curvature');

drawnow;

end

[res,~,~] = FastMarch(res,mindist,1,nb);

function ngradspeed = GVF(gradspeed, mu,dims)

r=dims(1);

c=dims(2);

slc = r\*c;

[Y,X] = meshgrid(1:c,1:r);

Y = Y(:);

X = X(:);

gs\_mag\_squared=sum(gradspeed.\*gradspeed);

node = [1:slc]';

rws = [reshape(repmat(node',[5,1]),[5\*slc,1])];

cols = rws + [repmat([0;-1;1;-r;r],[slc,1])];

% rws defines the row indices, cols defines the column indices.

s = repmat([0;-.25\*mu;-.25\*mu;-.25\*mu;-.25\*mu],[slc,1]);

first\_col= upsample(mu+(1-mu)\*gs\_mag\_squared,5)';

s=s+first\_col;

% s defines the value of each nonzero entry in the matrix. We initialize the off diagonal elements as

% the values for type (b) voxels but we need to change these for types (a) and (c).

I = zeros(slc\*5,1);

% We will use I to mark which rows we want to remove for type (a) and (c) pixels.

% bx = zeros(slc,1);

% by = zeros(slc,1);

% % Finally, bx and by will define the ‘b’ vector in our equation A\*z=b.

% dnode = q;

% % These are the node indices for our type (a) pixels.

% bx(dnode) = (1-mu)\*gs\_mag\_squared.\*gradspeed(1,:);

% by(dnode) = (1-mu)\*gs\_mag\_squared.\*gradspeed(2,:);

bx = [(1-mu)\*gs\_mag\_squared.\*gradspeed(1,:)]';

by = [(1-mu)\*gs\_mag\_squared.\*gradspeed(2,:)]';

% We want to mark for removal the off diagonal elements for these nodes since we have their exact

% value.

% I(reshape(repmat(5\*(dnode'-1)+1,[1,4])+repmat([1:4],[length(dnode),1]),[4\*length(dnode),1]))=1;

% Now we handle the boundary conditions.

N = find(X(:)==1);

I((N-1)\*5+2)=1;

I((N-1)\*5+4)=1;

I((N-1)\*5+5)=1;

% s((N-1)\*5+3)=-1;

N = find(X(:)==r);

I((N-1)\*5+3)=1;

I((N-1)\*5+4)=1;

I((N-1)\*5+5)=1;

% s((N-1)\*5+2)=-1;

N = find(Y(:)==1);

I((N-1)\*5+2)=1;

I((N-1)\*5+3)=1;

I((N-1)\*5+4)=1;

% s((N-1)\*5+5)=-1;

N = find(Y(:)==c);

I((N-1)\*5+2)=1;

I((N-1)\*5+3)=1;

I((N-1)\*5+5)=1;

% s((N-1)\*5+4)=-1;

rws = rws(~I(:));

cols = cols(~I(:));

s = s(~I(:));

% Now we construct sparse matrix A and solve A\*x=bx and A\*y=by

A = sparse(rws,cols,s,slc,slc);

x = A\bx;

y = A\by;

ngradspeed=[x y]';

function [kappa,ngrad,grad]=Curvature2(img,nb)

nodes=nb.q(1,1:nb.len);

kappa=[];

grad=[];

ngrad=[];

for i\_nodes = 1: length(nodes)

node=nodes(i\_nodes);

grad\_node=0.5\*Gradient(img,node);

ngrad\_node=sqrt(sum(grad\_node.\*grad\_node));

hess=Hessian(img,node);

kappa\_node=(grad\_node' \* hess \* grad\_node - ngrad\_node^2 \* trace(hess))/(2\*ngrad\_node^3);

kappa=[kappa kappa\_node];

grad=[grad grad\_node];

ngrad=[ngrad ngrad\_node];

end

function grad=Gradient(img, nodes)

global Edges

grad=[];

for i\_nodes=1:length(nodes)

node=nodes(i\_nodes);

neibs=Edges(node,:);

%% calculate gradient in x direction

if neibs(1)&& neibs(2)

grad\_x=img(neibs(1))-img(neibs(2));

end

if neibs(1)==0

grad\_x=img(node)-img(neibs(2));

end

if neibs(2)==0

grad\_x=img(neibs(1))-img(node);

end

%% calculate gradient in y direction

if neibs(3)&& neibs(4)

grad\_y=img(neibs(3))-img(neibs(4));

end

if neibs(3)==0

grad\_y=img(node)-img(neibs(4));

end

if neibs(4)==0

grad\_y=img(neibs(3))-img(node);

end

grad=[grad [grad\_x grad\_y]'];

end

function hess=Hessian(img,node)

global Edges

[r,c]=size(img);

neibs=Edges(node,:);

L=neibs(1);

R=neibs(2);

D=neibs(3);

U=neibs(4);

%% calculate dderiv\_x,dderiv\_y, dderiv\_xy

if L&&R

dderiv\_x=img(R)-2\*img(node)+img(L);

end

if L==0

dderiv\_x=img(R)-2\*img(node)+img(node);

end

if R==0

dderiv\_x=img(node)-2\*img(node)+img(L);

end

if D&&U

dderiv\_y=img(D)-2\*img(node)+img(U);

end

if D==0

dderiv\_y=img(node)-2\*img(node)+img(U);

end

if U==0

dderiv\_y=img(D)-2\*img(node)+img(node);

end

RU=R-1;

if RU<0

RU=node;

end

LU=L-1;

if LU<0

LU=node;

end

RD=R+1;

if RD>r\*c

RD=node;

end

LD=L+1;

if LD>r\*c

LD=node;

end

dderiv\_xy=1/4\*(RD-RU-LD+LU);

hess=[dderiv\_x dderiv\_xy;dderiv\_xy dderiv\_y];

function heap=HeapInit2(initlen)

if nargin<1

initlen=1000;

end

heap=struct;

heap.q=[-1\*ones(1,initlen); 3e8\*ones(1,initlen)];

heap.len=0;

function HeapInsert2(node,dist)

global heap;

% if the tree root is at index 1,

% with valid indices 1 through n,

% then each element a at index i has

% children at indices 2i and 2i +1

% its parent at index floor(i ? 2).

for i\_node=1:length(node)

heap.len=heap.len+1;

heap.q(:,heap.len)=[node(i\_node),dist(i\_node)]';

curr\_ind=heap.len;

parent\_ind=floor(heap.len/2);

while parent\_ind>0

if heap.q(2,parent\_ind)< heap.q(2,curr\_ind)

break

else

temp=heap.q(:,parent\_ind);

heap.q(:,parent\_ind)=heap.q(:,curr\_ind);

heap.q(:,curr\_ind)=temp;

curr\_ind=parent\_ind;

parent\_ind=floor(curr\_ind/2);

end

end

end

function [root\_node, root\_value]=HeapPop2()

global heap

if heap.len==0

root\_node=[];

root\_value=[];

return

end

root\_node=heap.q(1,1);

root\_value=heap.q(2,1);

last\_element=heap.q(:,heap.len);

heap.len=heap.len-1;

if heap.len==0

return

end

heap.q(:,1)=last\_element;

curr\_ind=1;

while 2\*curr\_ind<=heap.len

if 2\*curr\_ind+1<=heap.len % element has two childs

if heap.q(2,curr\_ind)< heap.q(2,2\*curr\_ind) && heap.q(2,curr\_ind)< heap.q(2,2\*curr\_ind+1)

break

else

% get the minimum of childs

ind\_childs=[2\*curr\_ind,2\*curr\_ind+1];

[~,ind\_min]=min(heap.q(2,ind\_childs));

temp=heap.q(:,ind\_childs(ind\_min));

heap.q(:,ind\_childs(ind\_min))=heap.q(:,curr\_ind);

heap.q(:,curr\_ind)=temp;

curr\_ind=ind\_childs(ind\_min);

end

else

if heap.q(2,curr\_ind)< heap.q(2,2\*curr\_ind+1)

break

else

temp=heap.q(:,2\*curr\_ind+1);

heap.q(:,2\*curr\_ind+1)=heap.q(:,curr\_ind);

heap.q(:,curr\_ind)=temp;

curr\_ind=2\*curr\_ind+1;

end

end

end