EECE 8395: Medical Image Segmentation

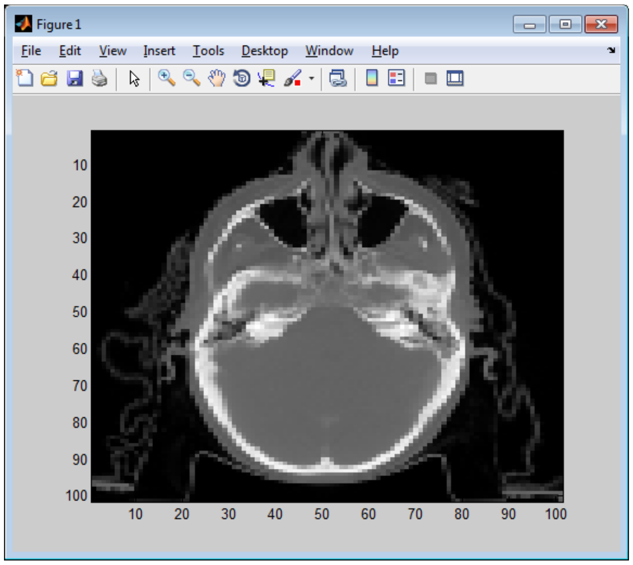
Vanderbilt University

Prof. Jack H. Noble

Livewire 2D

In this demo, we will see how to define edgecosts for the livewire method and how the livewire user interface is set up. First we load and display our image.

```
img = ReadNrrd('0522c0001\img.nrrd');
slcim = img;
fp = [180,140,1];
sp = [340,340,107];

slcim.data = img.data(fp(1):2:sp(1),fp(2):2:sp(2),fp(3):sp(3));
slcim.dim = size(slcim.data);
slc = slcim.data(:,:,79);
slcim.data = slcim.data/10+100;
figure(1); close(1); figure(1); clf; colormap(gray(256));
DisplayVolume(slcim,3,79);
```



We will be calling LiveWireSegment with the image and edgecost as input arguments:

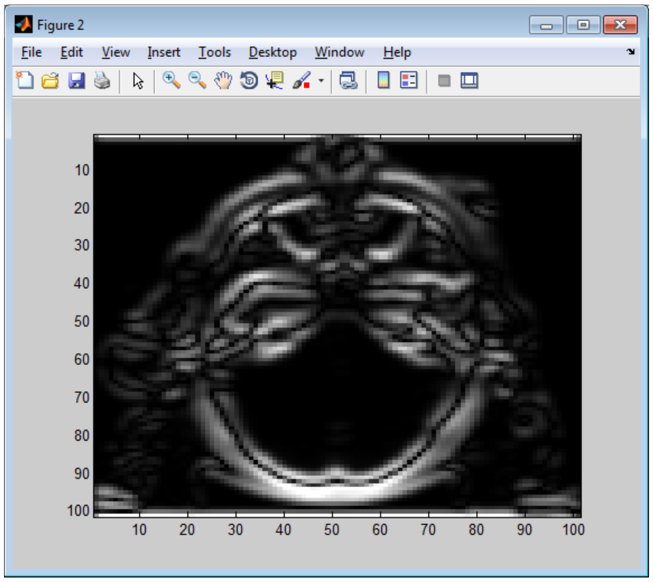
```
% will do >> cntr = LiveWireSegment(slc,edgecost);
% Setting up edgecost: costs for neighbor connections in the 8-connected
% neighborhood
%
% Edges(1) = x  , y+1;
% Edges(2) = x  , y-1;
% Edges(3) = x+1, y;
% Edges(4) = x-1, y;
% Edges(5) = x+1, y-1;
% Edges(6) = x-1, y+1;
% Edges(7) = x-1, y-1;
% Edges(8) = x+1, y+1;
```

So we need to define costs for these edges. To do that we will use edge detection filters. First, we apply a bit of smoothing to the image to reduce noise.

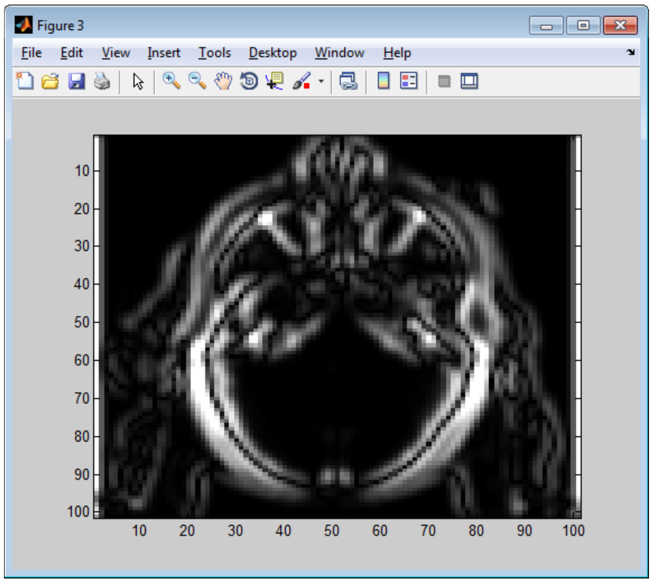
```
g = fspecial('gaussian',[5,5],1);
im2 = conv2(slc,g,'same');
```

Next we compute sobel filtering results on the smoothed image:

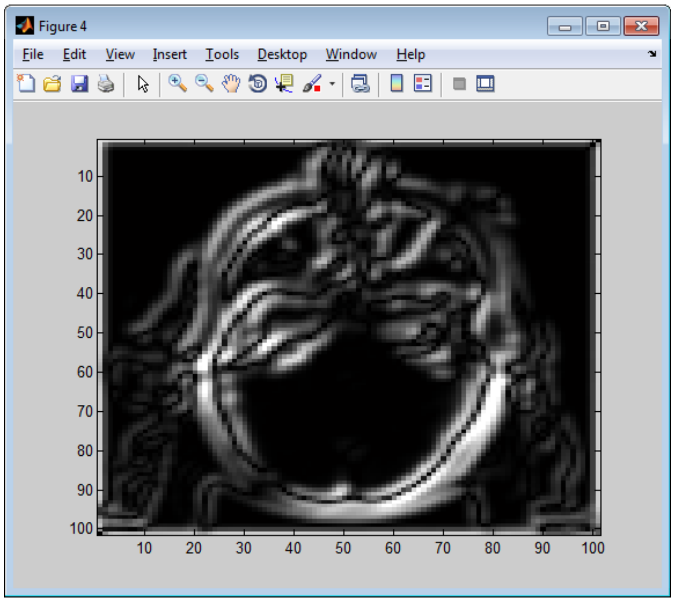
```
edgecost(:,:,1) = (abs(conv2(im2,[-1 0 1;-1 0 1;-1 0 1],'same')));
edgecost(:,:,2) = edgecost(:,:,1);
figure(2); clf; colormap(gray(256));
image(edgecost(:,:,1)'/10)
```



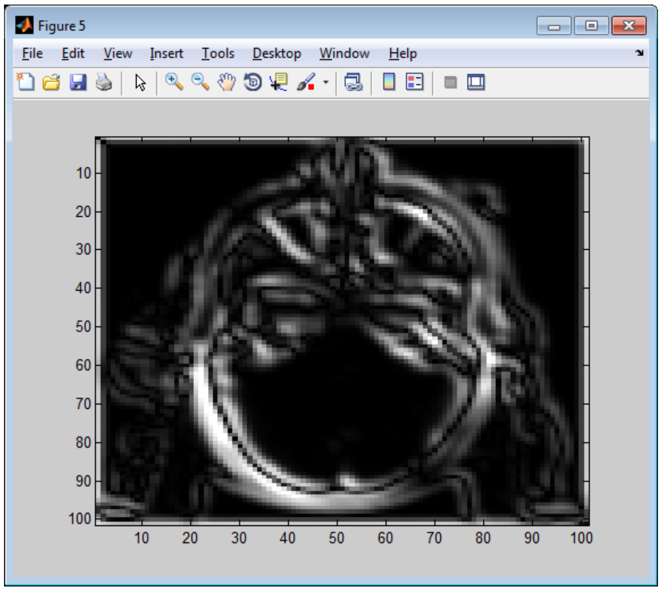
```
edgecost(:,:,3) = (abs(conv2(im2,[-1 0 1;-1 0 1;-1 0 1]','same')));
edgecost(:,:,4) = edgecost(:,:,3);
figure(3); clf; colormap(gray(256));
image(edgecost(:,:,3)'/10)
```



```
edgecost(:,:,5) = (abs(conv2(im2,sqrt(2)/2*[-2 -1 0;-1 0 1;0 1 2],'same')));
edgecost(:,:,6) = edgecost(:,:,5);
figure(4); clf; colormap(gray(256));
image(edgecost(:,:,5)'/10)
```

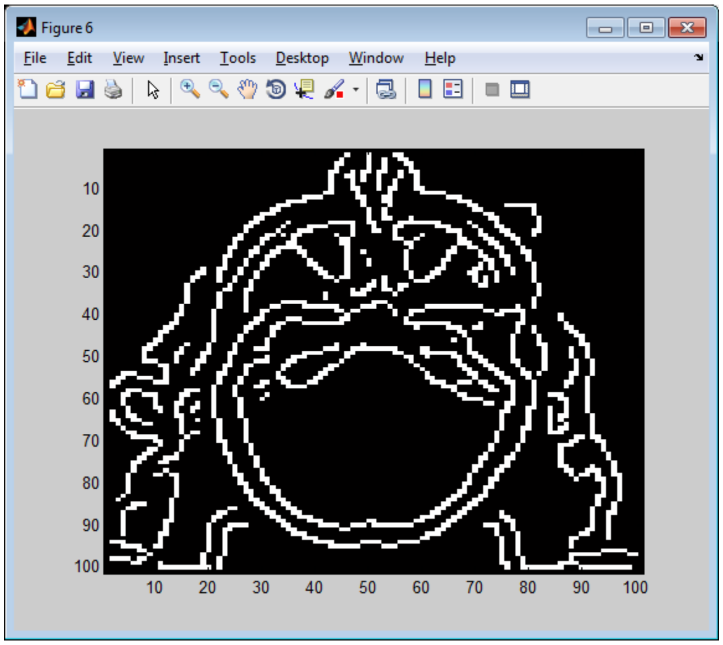


```
edgecost(:,:,7) = (abs(conv2(im2,sqrt(2)/2*[0 1 2;-1 0 1;-2 -1 0],'same')));
edgecost(:,:,8) = edgecost(:,:,7);
figure(5); clf; colormap(gray(256));
image(edgecost(:,:,7)'/10)
```



In addition to Sobel filtering, we can add a term based on the Canny edge detection method

```
help edge
Canny Method
     ---------------------------------
     BW = edge(I,'canny') specifies the Canny method.

    BW = edge(I,'canny',THRESH) specifies sensitivity thresholds for the
    Canny method. THRESH is a two-element vector in which the first element
    is the low threshold, and the second element is the high threshold. If
    you specify a scalar for THRESH, this value is used for the high
    threshold and 0.4*THRESH is used for the low threshold. If you do not
    specify THRESH, or if THRESH is empty ([]), edge chooses low and high
    values automatically.

BW = edge(slc,'canny',.1);
figure(6); clf; colormap(gray(256));
image(BW'*255)
```



As you can see, the canny method gives us a binary segmentation of the strong edges in the image. We next add it to the edgecost matrix with equivalent strength as the maximum Sobel filter response.

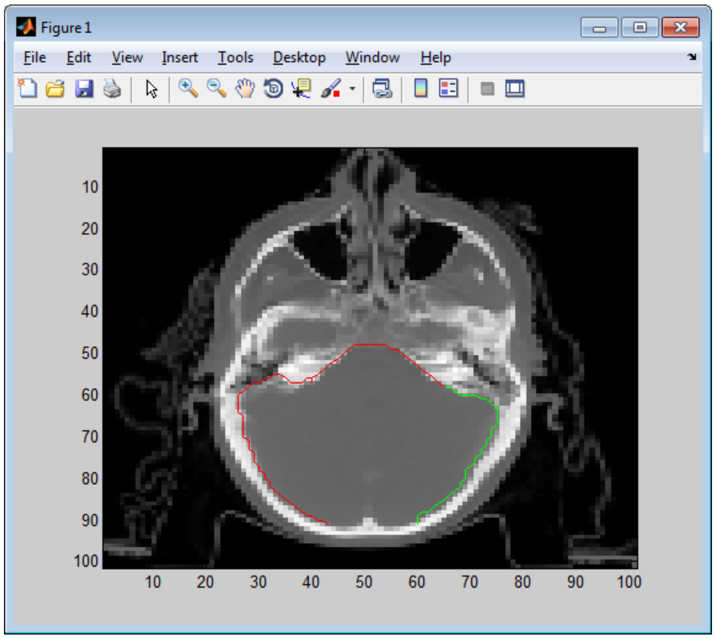
```
v = max(edgecost(:));
for i=1:8
    edgecost(:,:,i) = edgecost(:,:,i) + v*BW;
end
```

Now edgecost contains high values for strong edges, whereas we want to define a low cost at strong edges, so we invert it.

```
edgecost = max(edgecost(:)) - edgecost;
edgecost(:,:,5:8) = edgecost(:,:,5:8)*sqrt(2);
```

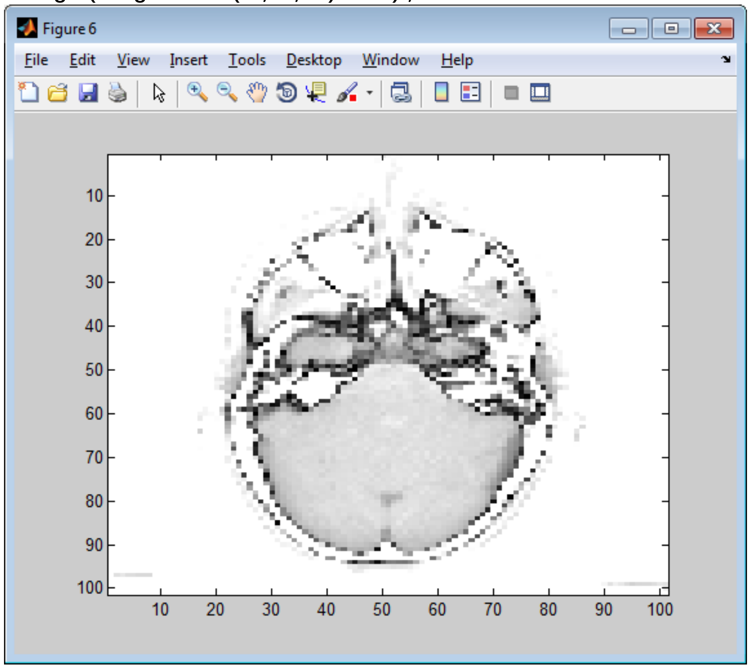
And now we can call our livewire function

```
figure(1); cntr = LiveWireSegment(slcim,edgecost,3,79);
```

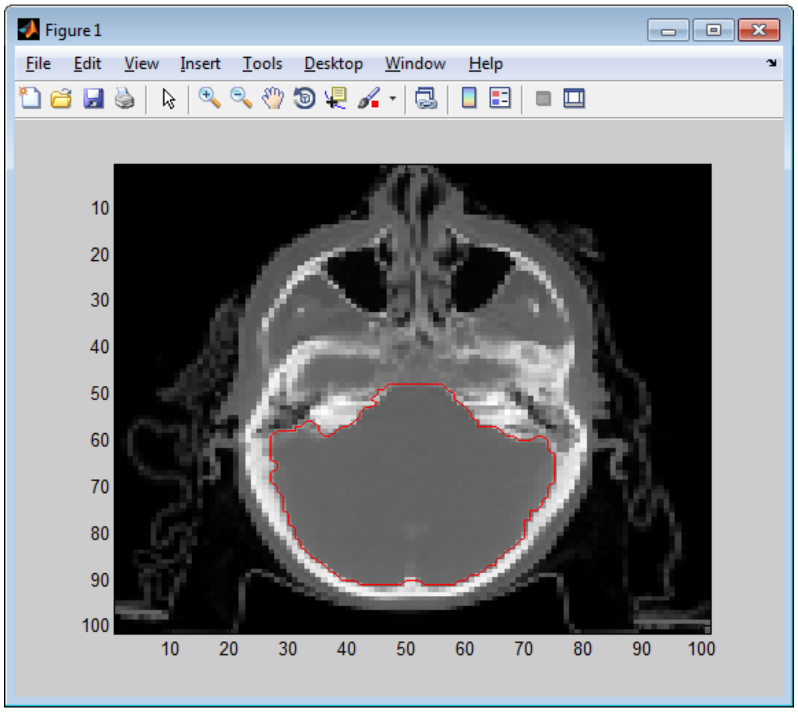


Note that the contour is imperfect, we have some errors around the lower right edge, especially. This is because the canny edge detector gave us false positives along this path. It is also possible to customize our cost function for this particular application. Here we want to separate bone and soft tissue. 500 HU should be a good intensity level to separate the two:

```
edgecost = repmat(abs(slc-500),[1,1,8]);
edgecost(:,:,5:8) = edgecost(:,:,5:8)*sqrt(2);
figure(6); clf; colormap(gray(256));
image(edgecost(:,:,1)'/2);
```



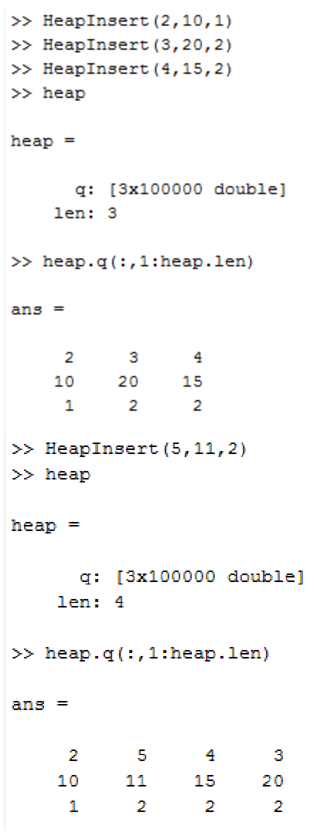
```
figure(1); close(1); figure(1); colormap(gray(256));
DisplayVolume(slcim,3,79);
cntr = LiveWireSegment(slcim,edgecost,3,79);
```



Now let’s look at the functions being used. Let’s start with the Heap for the graph search. We have three functions: `HeapInit`, `HeapInsert`, `HeapPop`. `HeapInit(initlen)`creates a global struct called `heap` that has two member variables – the heap data structure, which should be a 3 x `initlen` matrix and a ‘length’ variable, which keeps track of where the current end of the heap is.

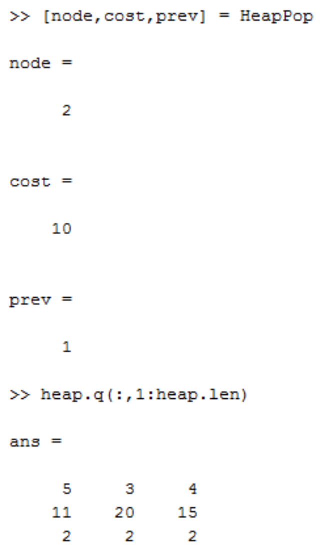
```
>> HeapInit
>> global heap
>> heap

heap =

      q: [3x100000 double]
    len: 0
```

`HeapInsert(node,cost,prev)` inserts the new node into the heap.



```
>> HeapInsert(2,10,1)
>> HeapInsert(3,20,2)
>> HeapInsert(4,15,2)
>> heap

heap =

      q: [3x100000 double]
    len: 3

>> heap.q(:,1:heap.len)

ans =

     2     3     4
    10    20    15
     1     2     2

>> HeapInsert(5,11,2)
>> heap

heap =

      q: [3x100000 double]
    len: 4

>> heap.q(:,1:heap.len)

ans =

     2     5     4     3
    10    11    15    20
     1     2     2     2
```

`[node,cost,prev] = HeapPop()` works like this:



```
>> [node,cost,prev] = HeapPop

node =

     2


cost =

    10


prev =

     1

>> heap.q(:,1:heap.len)

ans =

     5     3     4
    11    20    15
     2     2     2
```

Now let’s look at the helper functions for GraphSearch. Our GraphSearch call will look like:
`[nodepath,cost] = GraphSearch(@EdgeFunc,@NodeMark,@IsNodeMarked,@SetPointer,@GetPointer,seed,endnode)`
where the capitalized variables are function handles. We use handles because we want to be able to define these functions differently in later projects. `[Edges,EdgeCost] = EdgeFunc(indx)`

takes as input a node and returns its list of neighboring nodes in `Edges` and the cost of those edge connection in `EdgeCost`.

```
function [Edges,EdgeCost] = EdgeFunc(node)
global Edgs EdgCosts Edg_lens;
Edges = Edgs(node,1:Edg_lens(node));
EdgeCost = EdgCosts(node,1:Edg_lens(node));
return;
```

The function `NodeMark(node)` takes as input a node and marks it

```
function NodeMark(node)
global Done;
Done(node) = 1;
return;
```

The function `res = IsNodeMarked(node)`takes as input a node and returns whether the node is marked.

```
function res = IsNodeMarked(node)
global Done;
res = Done(node);
return;
```

The function `SetPointer(node,prev)` takes as input a node and its parent node to record the backpointer:

```
function SetPointer(node,prev)
global Pointer;
Pointer(node) = prev;
return;
```

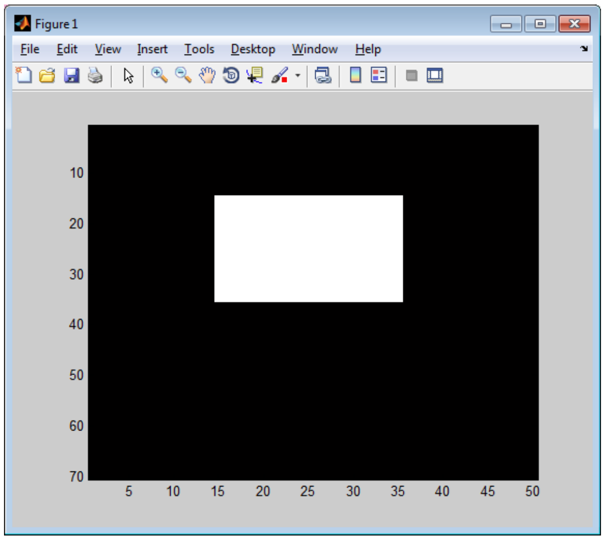
The function `prev = GetPointer(node)` takes as input a node and returns its backpointer:

```
function prev = GetPointer(node)
global Pointer;
prev = Pointer(node);
return;
```

Ok so let’s look at an example

```
global Done Pointer Edgs EdgCosts Edg_lens r c;

r=70;c=50;
testimg = zeros(r,c);
testimg(15:35,15:35) = 255;


figure(1); close(1); figure(1); colormap(gray(256));
```

```
image(testimg);
hold on;
```



Now we initialize our empty Done and Pointer lists.

```
Done = zeros(r*c,1);
Pointer = zeros(r*c,1);
```

Next we define our Edgs and EdgCosts data structures. Recall our Edge directions:

```
% Edges(1) = x, y+1;
% Edges(2) = x, y-1;
% Edges(3) = x+1, y;
% Edges(4) = x-1, y;
% Edges(5) = x+1, y-1;
% Edges(6) = x-1, y+1;
% Edges(7) = x-1, y-1;
% Edges(8) = x+1, y+1;
```

Our nodes are the pixels of the image. We can number them nodes = [1:r*c]. With this numbering, node +/- 1 are the nodes lying at x +/- 1; and node +/r are the neighbors at y +/- 1. Thus, we could try to define our edges like this:

```
Edgs = (repmat([1:r*c]',[1,8]) + repmat([r,-r,1,-1,1-r,r-1,-1-
r,1+r],[r*c,1]));
```

This gives us an [r*c , 8] matrix of edges. The problem is that it does not account for boundary conditions. Nodes on an image boundary have only 5 edges, rather than 8. Nodes on an image corner have only 3 edges. To account for boundary conditions, first we use meshgrid to get x,y positions for each node:

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

Then we define Edgs as:

```
Edgs =
[Y<c,Y>1,X<r,X>1,X<r&Y>1,X>1&Y<c,X>1&Y>1,X<r&Y<c].*(repmat([1:r*c]',[1,8]) +
repmat([r,-r,1,-1,1-r,r-1,-1-r,1+r],[r*c,1]));
```

This will result in zeros in Edgs for edges that do not exist due to boundary conditions. We can define the edge cost as follows:

```
EdgCosts = repmat(1+testimg(1:r*c)',[1,8]);
EdgCosts(:,5:8) = EdgCosts(:,5:8)*sqrt(2);
```

Thus, our cost function is based on the input image. Note we add 1 to avoid having zero cost edges. Also note we multiply the diagonal edges by sqrt(2) because they are longer edges. We can find the total number of valid edges for each node and clean up the zero edges as follows:

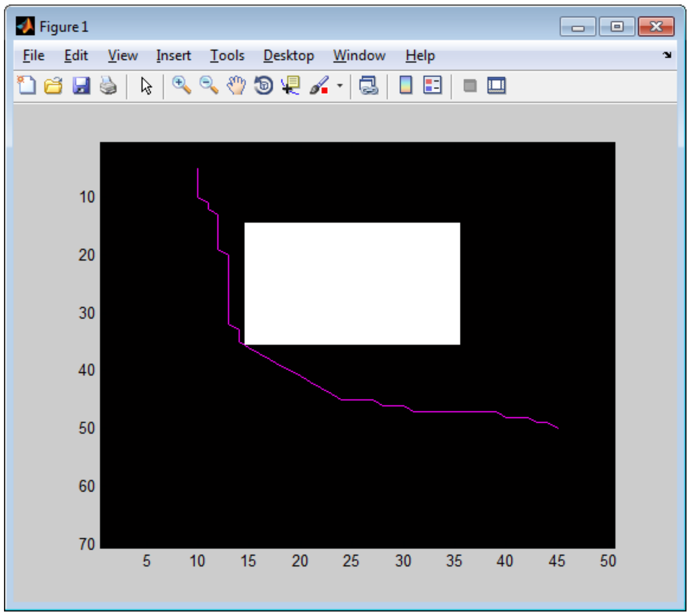
```
Edg_lens = sum(Edgs'~=0)';
for i=1:r*c
    msk = Edgs(i,:)>0;
    Edgs(i,1:Edg_lens(i)) = Edgs(i,msk);
    EdgCosts(i,1:Edg_lens(i)) = EdgCosts(i,msk);
end
```

All that’s left is to define a seed and endnode. This defines a seed at x=10,y=5 and endnode at x=45,y=50:

```
seednode = (5-1)*r+10;
endnode =  (50-1)*r+45;
```

Now we compute the minimum cost path and display it:

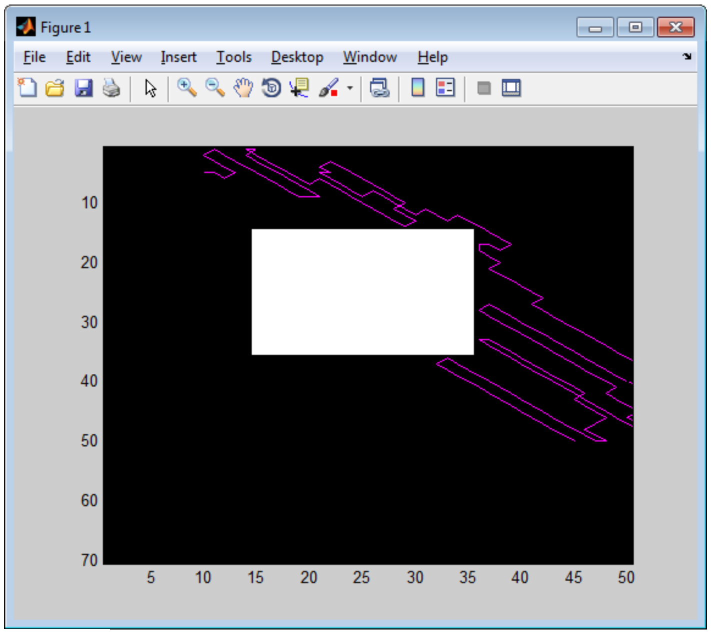
```
[nodepath,cost] =
GraphSearch(@EdgeFunc,@NodeMark,@IsNodeMarked,@SetPointer,@GetPointer,seednod
e,endnode);
pnts = [mod(nodepath'-1,r)+1,floor((nodepath'-1)/r)+1];
plot(pnts(:,1),pnts(:,2),'m')
```



```
cost =

   68.8701
```

Makes sense! What happens if we don't add '1' to the image in the cost function?:

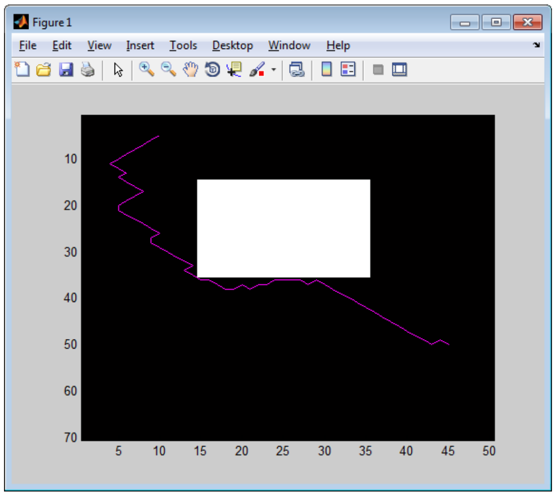
```
EdgCosts = repmat(testimg(1:r*c)',[1,8]);
```



```
cost =

     0
```

What happens if we don't use the square root of 2 factor on the diagonal edges in the cost function:

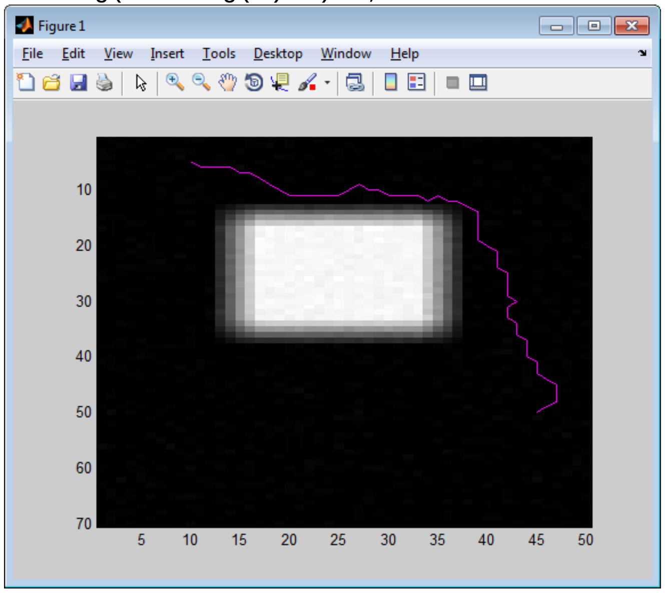
```
% EdgCosts(:,5:8) = EdgCosts(:,5:8)*sqrt(2);
```



```
cost =

    61
```

How about if our image is blurry and noisy?

```
testimg = conv2(testimg,fspecial('gaussian',[5,5],3),'same') +
3*randn(size(testimg));
testimg(testimg(:)<0)=0;
```



```
cost =

   79.3367
```

OK, so we have our minimum cost path method working. Now let’s look at how we are actually designing the callback functions. First we can set a mouse button down function like this:

```
iptaddcallback(hfig,'WindowButtonDownFcn',@MouseButtonDownCallback);
```

If we want the callback to initiate the Livewire process, `MouseButtonDownCallback` looks like this:

```
function MouseButtonDownCallback(h,e)
global c r

a = get(h,'SelectionType');
if ~strcmp(a,'normal')
    return;
end

a = get(gca,'CurrentPoint');
x = round(a(1,1));
y = round(a(1,2));
if ((x>=1)&&(y>=1)&&(x<=r)&&(y<=c))
   GraphSearch(@EdgeFunc,@NodeMark,@IsNodeMarked,@SetPointer,@GetPointer,x+(y
      -1)*r,-1);
end
```

And our mouse move callback looks like this:

```
iptaddcallback(hfig,'WindowButtonMotionFcn',@MouseMoveCallback);
```

And then the mouse move callback function is:

```
function MouseMoveCallback(h,e)
global r c;

a = get(gca,'CurrentPoint');
x = round(a(1,1));
y = round(a(1,2));


node = round([x;y]);
if ~((node(1)>=1)&&(node(2)>=1)&&(node(1)<=r)&&(node(2)<=c))
    return;
end
path(:,1) = node;
pathcnt = 1;

prev=GetPointer((node(2)-1)*r+node(1));
while prev~=0
    pathcnt = pathcnt+1;
    path(:,pathcnt) = [mod((prev-1),r), floor((prev-1)/r)]+1;
    prev=GetPointer(prev);
end


hold off
DisplayVolume();
title('LiveWire active');
```

```
hold on;
plot(path(1,1:pathcnt),path(2,1:pathcnt),'g');

drawnow;
return;
```

Our DisplayVolume function looks like this:

```
function DisplayVolume(img,direction,slc)
if nargin<2
    direction=3;
end
if nargin<3 && nargin>0
    slc = floor(img.dim(direction)/2);
end
hfig = gcf;
if nargin>0
    clf;
    set(hfig,'KeyPressFcn','');


    inp.img = img;
    inp.slc = slc;
    inp.direction = direction;
    guidata(hfig,inp);
    iptaddcallback(hfig,'KeyPressFcn',@KeyboardCallback);
else
    inp = guidata(hfig);
    if ~isfield(inp,'img')
        return;
    end
end

if inp.direction==3
    image(inp.img.data(:,:,inp.slc)');
    daspect([inp.img.voxsz(2) inp.img.voxsz(1) 1]);
    xlabel('x');
    ylabel('y');
    z = 'z';
elseif inp.direction==2
    axis([1,inp.img.dim(1),1,inp.img.dim(3)]);
    hold on;
    image((squeeze(inp.img.data(:,inp.slc,:))'));
    daspect([inp.img.voxsz(3) inp.img.voxsz(1) 1]);
    xlabel('x');
    ylabel('z');
    z = 'y';
else
    axis([1,inp.img.dim(2),1,inp.img.dim(3)]);
    hold on;
    image((squeeze(inp.img.data(inp.slc,:,:))'));
    daspect([inp.img.voxsz(3) inp.img.voxsz(2) 1]);
    xlabel('y');
    ylabel('z');
    z = 'x';
end
title(['Slice ',z,' = ',num2str(inp.slc)]);

if isfield(inp,'cntrs')
    hold on

plot(inp.cntrs(1,1:inp.lcntrs(inp.slc),inp.slc),inp.cntrs(2,1:inp.lcntrs(inp.
slc),inp.slc),'r');
end

if nargin>0
drawnow;
end
return;
```

Let’s try it on the optic nerve.

```
% optic nerve
fp = [260,150,70];
sp = [320,220,107];

slcim.data = img.data(fp(1):sp(1),fp(2):sp(2),fp(3):sp(3))/2+100;
slcim.dim = size(slcim.data);
slc = slcim.data(:,:,19);

[r,c] = size(slc);
Done = zeros(r*c,1);
Pointer = zeros(r*c,1);

clear edgecost;

g = fspecial('gaussian',[5,5],1);
im2 = conv2(slc,g,'same');

edgecost(:,:,1) = (abs(conv2(im2,[-1 0 1;-1 0 1;-1 0 1],'same')));
edgecost(:,:,2) = edgecost(:,:,1);
edgecost(:,:,3) = (abs(conv2(im2,[-1 0 1;-1 0 1;-1 0 1]','same')));
edgecost(:,:,4) = edgecost(:,:,3);
edgecost(:,:,5) = (abs(conv2(im2,sqrt(2)/2*[-2 -1 0;-1 0 1;0 1 2],'same')));
edgecost(:,:,6) = edgecost(:,:,5);
edgecost(:,:,7) = (abs(conv2(im2,sqrt(2)/2*[0 1 2;-1 0 1;-2 -1 0],'same')));
edgecost(:,:,8) = edgecost(:,:,7);
BW = edge(slc,'canny',.1);

v = max(edgecost(:));
for i=1:8
    edgecost(:,:,i) = edgecost(:,:,i) + v*BW;
end
edgecost = max(edgecost(:)) - edgecost;
edgecost(:,:,5:8) = edgecost(:,:,5:8)*sqrt(2);
```

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

Edgs =
[Y<c,Y>1,X<r,X>1,X<r&Y>1,X>1&Y<c,X>1&Y>1,X<r&Y<c].*(repmat([1:r*c]',[1,8]) +
repmat([r,-r,1,-1,1-r,r-1,-1-r,1+r],[r*c,1]));
EdgCosts = reshape(edgecost,[r*c,8]);
EdgCosts(:,5:8) = EdgCosts(:,5:8)*sqrt(2);
Edg_lens = sum(Edgs'~=0)';
for i=1:r*c
    msk = Edgs(i,:)>0;
    Edgs(i,1:Edg_lens(i)) = Edgs(i,msk);
    EdgCosts(i,1:Edg_lens(i)) = EdgCosts(i,msk);
end

figure(1); close(1); hfig = figure(1); colormap(gray(256));
DisplayVolume(slcim,3,19);


iptaddcallback(hfig,'WindowButtonDownFcn',@MouseButtonDownCallback);
iptaddcallback(hfig,'WindowButtonMotionFcn',@MouseMoveCallback);
```

That lets us start one path but then we are stuck. Why? Our global Done vector needs to be reinitialized

```
function MouseButtonDownCallback(h,e)
global c r Done
…
   Done(:)=0;
 …
```

That’s not quite right either because it’s not recording the previous paths and we are losing that information. It’s also not great that we have to select the seedpoint before adding the mouse move callback.

```
function MouseMoveCallback(h,e)
global r c midcontour path pathcnt;

if ~midcontour
    return;
end
…
plot(donepath(1,1:donepathcnt),donepath(2,1:donepathcnt),'r');
…
```

```
function MouseButtonDownCallback(h,e)
global c r Done midcontour donepath donepathcnt path pathcnt

a = get(h,'SelectionType');
if ~strcmp(a,'normal')
    return;
end

if midcontour==0
    a = get(gca,'CurrentPoint');
    x = round(a(1,1));
    y = round(a(1,2));
    if ((x>=1)&&(y>=1)&&(x<=r)&&(y<=c))
        Done(:)=0;
        donepathcnt = 1;
        donepath(:,donepathcnt) = [x;y];
        GraphSearch(@EdgeFunc,@NodeMark,@IsNodeMarked,@SetPointer,@GetPointer
            ,x+(y-1)*r,-1);
        midcontour=1;
    end
else
    donepath(:,donepathcnt+1:donepathcnt+pathcnt-1) = path(:,pathcnt-1:-1:1);
    donepathcnt = donepathcnt + pathcnt-1;
    x = round(donepath(1,donepathcnt));
    y = round(donepath(2,donepathcnt));
    Done(:)=0;
     GraphSearch(@EdgeFunc,@NodeMark,@IsNodeMarked,@SetPointer,@GetPointer,(y-
        1)*r+x,-1);
end
return;

global midcontour
midcontour = 0;
figure(1); close(1); hfig = figure(1); colormap(gray(256));
DisplayVolume(slcim,3,19);
iptaddcallback(hfig,'WindowButtonDownFcn',@MouseButtonDownCallback);
iptaddcallback(hfig,'WindowButtonMotionFcn',@MouseMoveCallback);
```

The midcontour variable is set to 1 once the livewire begins and so before it is set, the mousemove callback just exits. Also, when the mouse is clicked and midcontour is already set to 1, we dump the current path into donepath. Note that by making path and pathcnt global, we don’t have to re-compute path in the mousebuttondown callback, we just use the last path that was computed in mousemovecallback. Also note that path is dumped into donepath backwards. This is necessary to keep the final path, which is the chain of individual paths, in order. Also note that the last entry in path is ignored because it is already at the end of donepath. Finally, we rerun the GraphSearch at the new seedpoint, which is the end of the last path. We also update our KeyboardCallback that is used in DisplayVolume so that when a contour is being drawn, we cannot page through the slices:

```
function KeyboardCallback(han,e)
global midcontour
if midcontour==1
```

```
    return;
end
h = guidata(han);
if (strcmp(e.Key,'uparrow'))
    h.slc=min(h.img.dim(h.direction),h.slc+1);
elseif (strcmp(e.Key,'downarrow'))
    h.slc=max(1,h.slc-1);
end
guidata(han,h);
DisplayVolume();

return;
```

That seems to be working well. The only thing we are missing is how do we stop?

```
function MouseButtonDownCallback(h,e)
…
    donepath(:,donepathcnt+1:donepathcnt+pathcnt-1) = path(:,pathcnt-1:-1:1);
    donepathcnt = donepathcnt + pathcnt-1;
    if norm(donepath(:,donepathcnt)-donepath(:,1))>2
        x = round(donepath(1,donepathcnt));
        y = round(donepath(2,donepathcnt));
        Done(:)=0;

        GraphSearch(@EdgeFunc,@NodeMark,@IsNodeMarked,@SetPointer,@GetPointer
        ,(y-1)*r+x,-1);
    else
        hold off;
        DisplayVolume();

        hold on;
        plot(donepath(1,1:donepathcnt),donepath(2,1:donepathcnt),'r');

        plot([donepath(1,donepathcnt),donepath(1,1)],[donepath(2,donepathcnt)
        ,donepath(2,1)],'r');
        drawnow;
        set(gcf,'WindowButtonMotionFcn','');
        set(gcf,'WindowButtonDownFcn','');

        midcontour=0;
    end
```

That seems to be working how we would like. How do we put this all inside a function that will return the contour?

```
function contour = LiveWireSegment(im_in, cost_in)
global done donepath donepathcnt

done=0;
…

while ~(done)
    pause(.5);
end
contour = donepath(:,1:donepathcnt);
```

```
function MouseButtonDownCallback(h,e)
global c r Done midcontour donepath donepathcnt path pathcnt done
…
done=1;
…
```

That works well!