**Report for Assignment 2 – Part IV: Segmentation by Deformable Models**

**Ravikiran Janardhana**

**Medical Image Analysis – COMP 775**

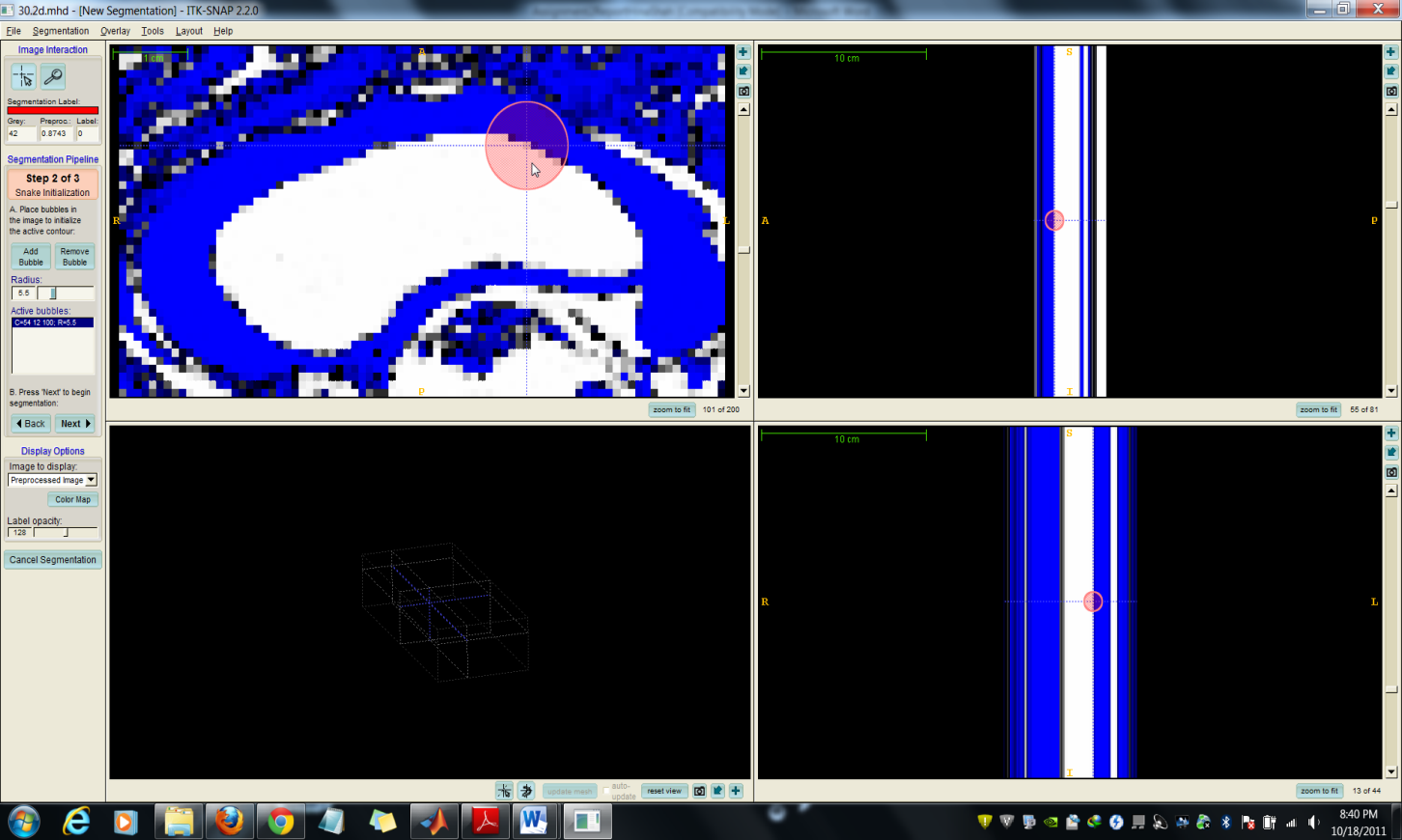
**Fall 2011**

**Tasks completed**

1. Implemented the Active Shape Model method to segment medical images.
2. Generated the segmentations using ITK-Snap which employs Geodesic Snakes.
3. Implemented the Dice Comparison.

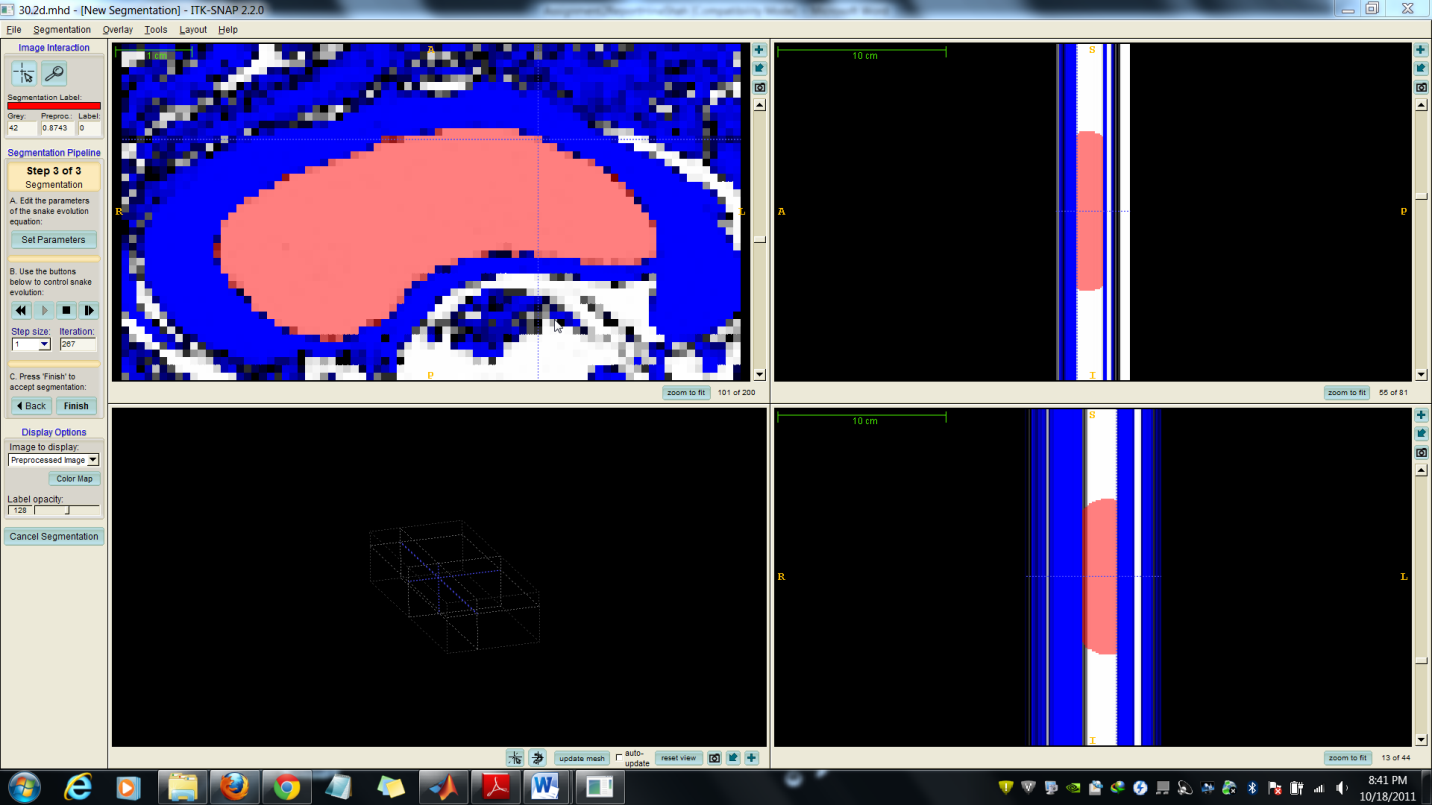
**SNAP Segmentation**

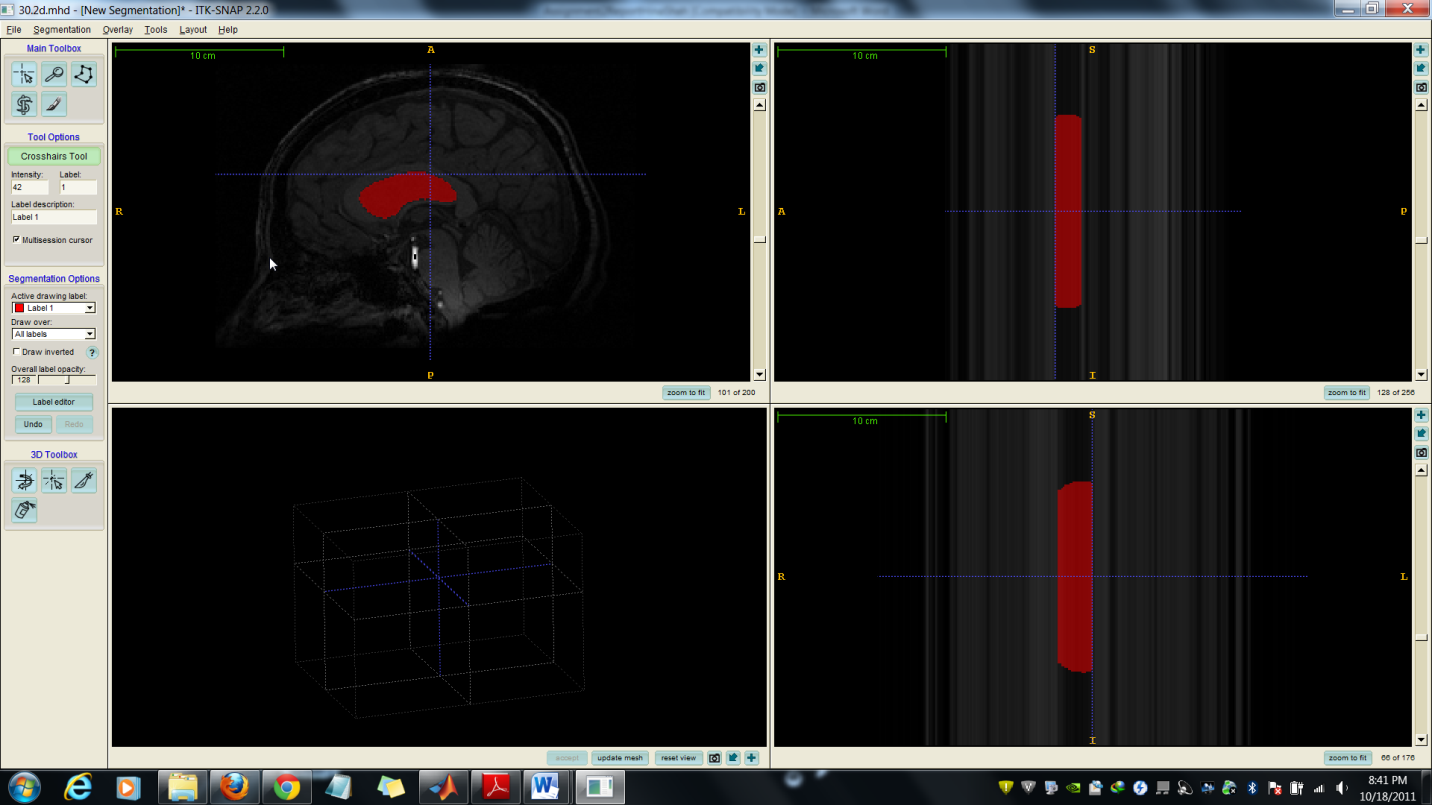
The target single MR slice images were piled up to generate multi slice MR images in order to segment it in ITK-SNAP. For different shapes, different values of forces were given in order to restrict the velocities to be within the boundaries. The initial balloon was sometimes kept such that a part of it is inside and a part of it is outside the boundary and below is a screenshot of one such initialization bubble. In most of the other cases, the balloon was kept at the center.

****

The preprocessing of the image was done via Intensity Regions by setting an upper threshold in between 40-60 (except for 93.2.mhd which required a threshold of 120) and smoothing it out.

The following screenshot shows the completed segmentation using ITKSnap:-





The different force values affecting the segmentation in SNAP are as below:-

a) Balloon force: if value greater than 0, tends to expand the boundary/snakes, else tends to contract. A large increase in the balloon force may cause the segmentation to leak out.

b) Curvature force: Tends to make the boundary smooth, and helps in stopping the leaking. A high value works perfectly for stopping leakages at boundaries and narrow regions.

Following is the table of different values and number of iterations required for each data:

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Balloon Force** | **Curvature Force** | **No. of Iterations** |
| 30.2d.mhd | 0.3 | 0.5 | 311 |
| 34.2d.mhd | 1.0 | 0.2 | 208 |
| 39.2d.mhd | 0.6 | 0.3 | 288 |
| 45.2d.mhd | 0.6 | 0.2 | 201 |
| 55.2d.mhd | 0.9 | 0.2 | 272 |
| 58.2d.mhd | 0.5 | 0.2 | 253 |
| 68.2d.mhd | 1.3 | 0.3 | 183 |
| 81.2d.mhd | 0.6 | 0.2 | 216 |
| 89.2d.mhd | 0.7 | 0.2 | 250 |
| 93.2d.mhd | 0.6 | 0.3 | 204 |

As seen from the above table, datasets 34.2d.hd and 68.2d.mhd required greater balloon force and this is due to the fact that the segmentation targets in these datasets were more bent and narrow at one end.

**Dice Program**

1) asmdice.m

% Compute Dice Measure of correct segmentation, snap segmentation and ASM

% segmentation

% Change folder directory to ITKSnap Segmentation results for the other part

files=dir('../asmSegImg/\*.mhd');

% For each file repeat the below

for k = 1: length(files)

asmFile = char(strcat('../asmSegImg/',files(k).name));

tempFile = regexp(files(k).name, '\.', 'split');

correctFile = char(strcat('../testBin/', ...

tempFile(1),'.seg.',tempFile(2),'.',tempFile(3)));

% Load the correct segmentation and the target segmentation via ASM

correctSeg = loadMETA(correctFile);

asmSeg = loadMETA(asmFile);

% Initialize values

correctArea = 0;

asmArea = 0;

commonArea = 0;

for i = 1 : size(correctSeg,1)

for j = 1 : size(correctSeg,2)

% Increment correctArea counter if value is not 0

if correctSeg(i,j) ~= 0

correctArea = correctArea + 1;

end

% Increment ASM counter if value is not 0

if asmSeg(i,j) ~= 0

asmArea = asmArea + 1;

end

% Increment commonArea if ASM and correct seg value is not 0

if correctSeg(i,j) ~= 0 && asmSeg(i,j) ~= 0

commonArea = commonArea + 1;

end

end

end

% Compute dice value and display

avg = (correctArea+asmArea)/2;

diceVal = commonArea / avg;

status = char(strcat('File = ',files(k).name, ...

' Dice = ',num2str(diceVal)));

disp(status);

end

2) writePDM.m

function writePDM( pdm, pdmFname, MRFname )

% Load original MR slice file and compute dimensions

I = loadMETA(MRFname);

temp = zeros(size(I,1), size(I,2));

%temp = transpose(temp);

maxx = size(temp,1);

maxy = size(temp,2);

% Store each possible point combination in points

for i = 1 : maxx

for j = 1 : maxy

points((i-1)\*maxy+j,1) = i;

points((i-1)\*maxy+j,2) = j;

end

end

% Check if these points lie inside the PDM polygon

result = pip2(points,round(pdm));

for i = 1 : size(result,1)

% Point lies inside polygon

if(result(i) ~= 0)

temp(points(i,1),points(i,2)) = 255;

end

end

% Write the file out as \*.mhd

writeMETA(temp,pdmFname);

imshow(transpose(temp));

end

3) pip2.m

function result = pip2(point, polygon, draw) % Jack Snoeyink

% point inside polygon test with simulation of simplicity and two outcome

% input: point=kx2, polygon=nx2, optional draw=false(default)

% output: logical k-vector with true(1) = inside, false(0) = outside

%

%% Test code

% polygon = round(rand(3,2).\*100);

% point = round(rand(1000,2).\*100);

% r=pip2(point, polygon, true);

%% Clean up input

if nargin < 3

draw = false;

end

if any(polygon(1,:) ~= polygon(end,:)) % convert polygon to closed polyline

polygon(end+1,:) = polygon(1,:);

end

segi = 1:(size(polygon, 1)-1); % first index for each segment

%% Loop over each point

n = size(point,1);

result = false(n,1); % default outside

for j = 1:n

x = point(j,1); % ray origin

y = point(j,2);

%% find the indices of segments that cross the ray (no horizontal segs do).

ci = segi((polygon(segi,2)<= y & polygon(segi+1,2) > y) | (polygon(segi,2) > y & polygon(segi+1,2) <= y));

%% count crossing if input x is strictly < interpolated x on segment.

X1 = polygon(ci+1,1);

Y1 = polygon(ci+1,2);

crossings = sum((x-X1) < (polygon(ci,1)-X1).\*(y-Y1)./(polygon(ci,2)-Y1));

result(j) = mod(crossings,2)==1;

end

%% plot

if draw

plot(polygon(:,1),polygon(:,2),'b-d', ... % polygon

point(result,1), point(result,2), 'ro', ... % inside

point(~result,1), point(~result,2), 'y+'); % outside

end

**Dice Results**

In order to compare ASM method and SNAP method segmentations, the center of gravity of mean PDM in ASM and the position of the initial bubble in SNAP were kept same.

The parameters passed to the Dice program is the segmented images from ASM/SNAP and the correctly segmented images. The following table shows the Dice measure for ASM/SNAP segmentations:-

|  |  |  |
| --- | --- | --- |
| **Data** | **Dice Measure of ASM  (max Iterations = 1000)** | **Dice Measure of SNAP (Geodesic Snakes)** |
| 30.2d.mhd | 0.84174 | 0.91704 |
| 34.2d.mhd | 0.62543 | 0.89376 |
| 39.2d.mhd | 0.81626 | 0.94599 |
| 45.2d.mhd | 0.82031 | 0.92282 |
| 55.2d.mhd | 0.78437 | 0.88372 |
| 58.2d.mhd | 0.83871 | 0.96317 |
| 68.2d.mhd | 0.68829 | 0.85632 |
| 81.2d.mhd | 0.82988 | 0.94783 |
| 89.2d.mhd | 0.81563 | 0.92496 |
| 93.2d.mhd | 0.88244 | 0.95757 |

From the above table it can be inferred that the average dice measure for ASM segmentation was **0.794306** and for SNAP (Geodesic Snakes) segmentation was **0.921318**.

It can be observed that both SNAP and ASM segmentation were not efficient while segmenting datasets **34.2d.mhd** and **68.2d.mhd**. In these two datasets, ASM suffered significantly greater when compared to Geodesic snakes used in ITKSnap. This can be attributed to the greater deviation of the segmentation target from the mean PDM in case of ASM and greater bending and narrowness at one end in case of SNAP.

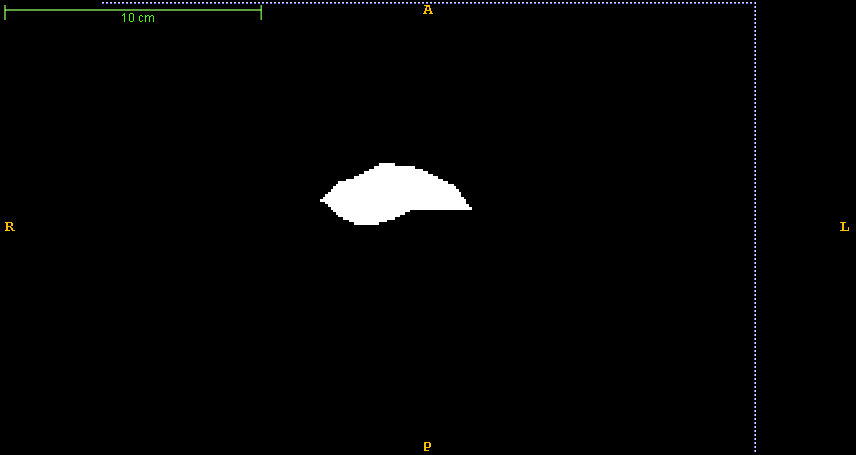
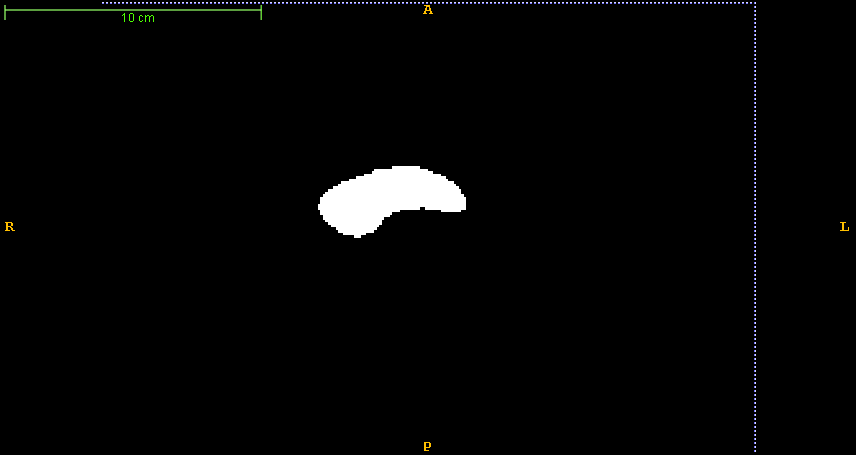
In ASM, the maximum number of iterations was fixed as 1000 when there was no convergence. It was observed that most of the datasets didn’t converge and thus were stopped after 1000 iterations. In some cases it was the observed that the shape space PDM was going in cycles (cyclic) as the iterations proceeded, thus not being able to converge.

Overall, it can be said that both ASM and ITKSnap (Geodesic Snakes) segmentations worked pretty well except when the target image was narrow at one/both ends.

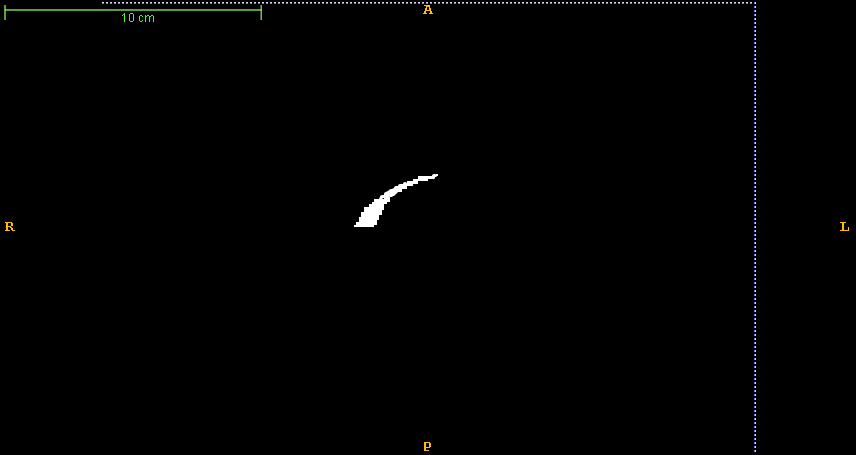
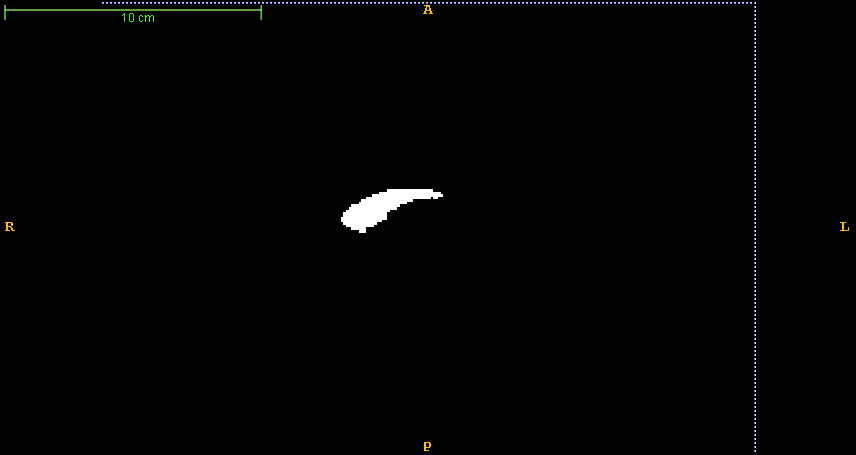
**Visual Comparison of ASM and ITKSnap (Geodesic Snakes) Segmentation**

All the 10 dataset segmentations are visually compared below, where the image on the left is obtained via ASM Segmentation and the image on the right is obtained via SNAP Segmentation which uses Geodesic Snakes.

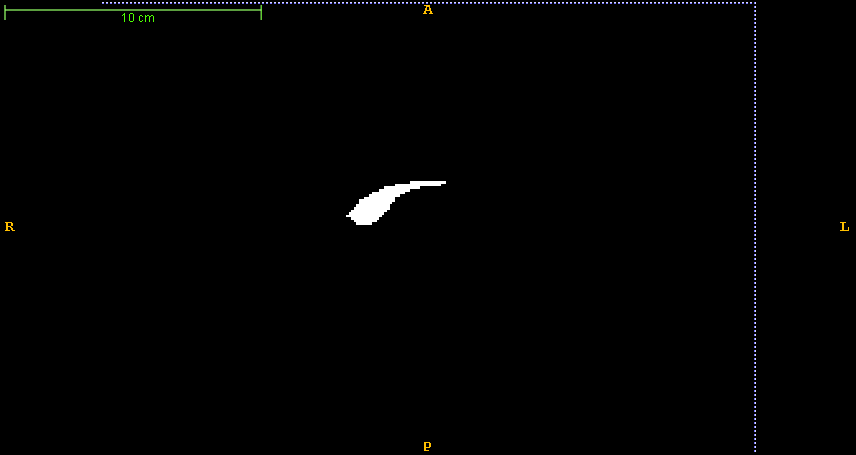
1) 30.2d.mhd

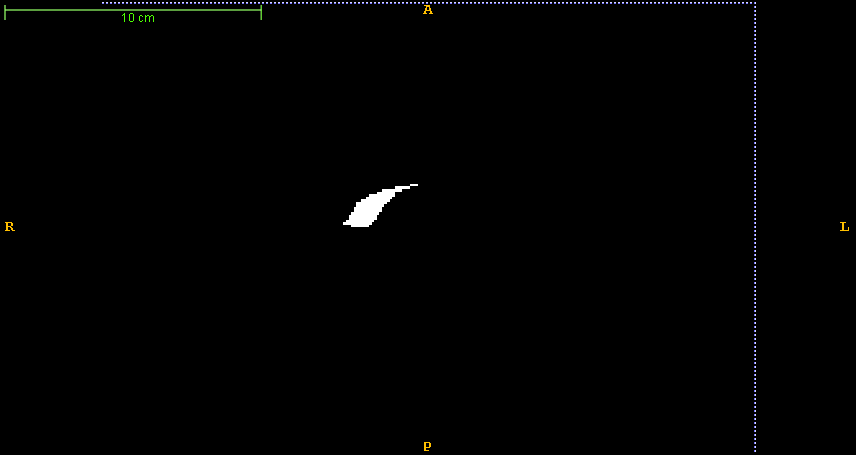
2) 34.2d.mhd

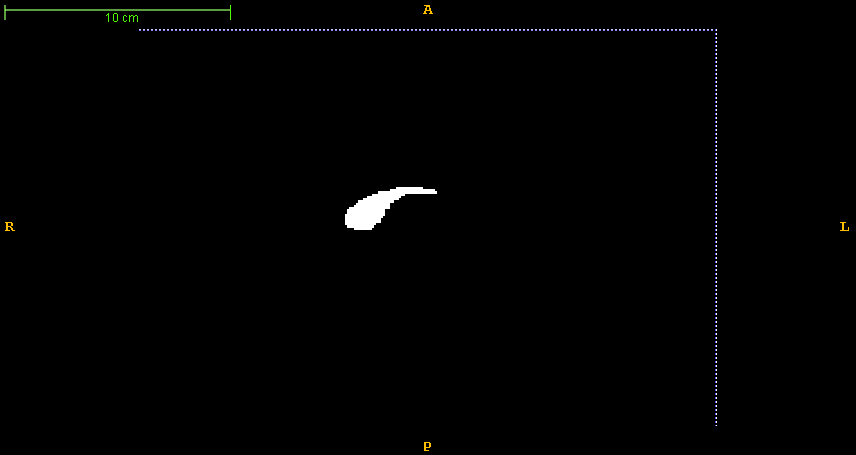
3) 39.2d.mhd

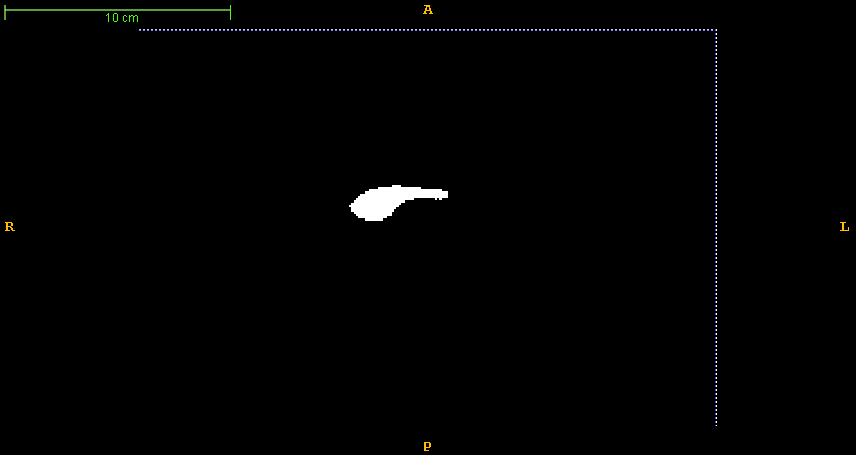
4) 45.2d.mhd

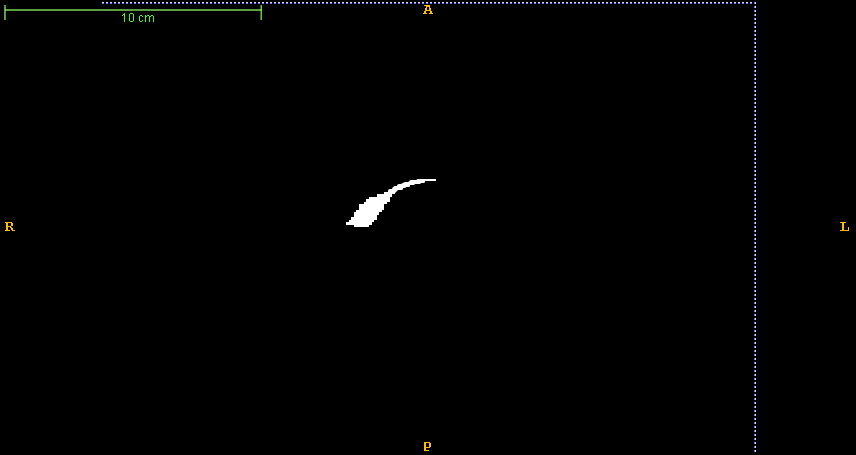
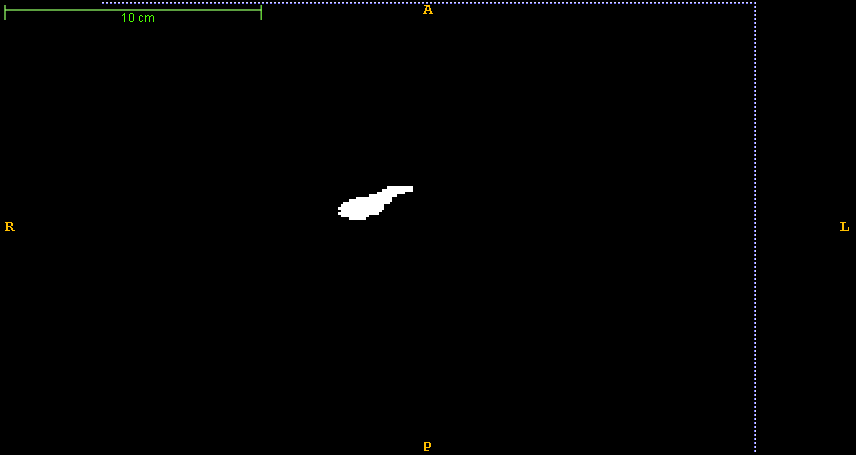
5) 55.2d.mhd

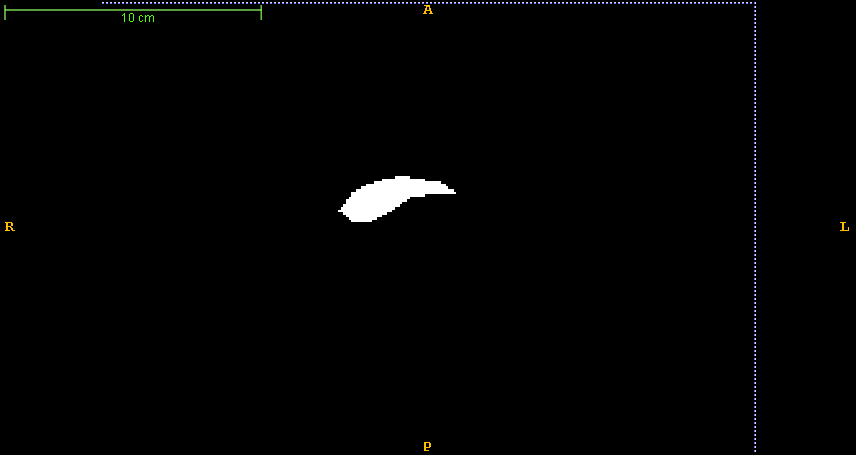
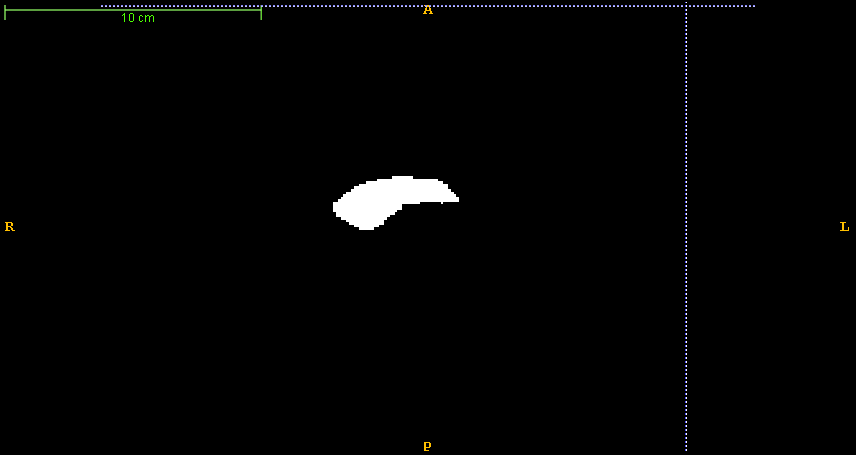
6) 58.2d.mhd

7) 68.2d.mhd

8) 81.2d.mhd

9) 89.2d.mhd

10) 93.2d.mhd

**ASM Code**

activeShapeModel.m (main program)

% Start reading the training PDMs

% Read all the points

% pts contains all the 21 point sets (21x68x2)

% (:,:,1) - contains x values

% (:,:,2) - contains y values

files=dir('../trainPDM/\*.pts');

for i = 1: length(files)

fName = strcat('../trainPDM/',files(i).name);

pts(i,:,:) = readPoints(fName);

end

% backup the original points read

orig\_pts = pts;

mean\_orig\_pts = squeeze(mean(pts));

% compute the mean of original points, used for translating back to mean

% and object space

mean\_x = mean(mean\_orig\_pts(:,1));

mean\_y = mean(mean\_orig\_pts(:,2));

% Compute Center of Gravity (Mean) of each point set and subtract each one

% of the points from the center of gravity of the same set

for i = 1 : size(pts, 1)

cg\_pts = squeeze(mean(pts(i,:,:)));

for j = 1 : size(pts, 2)

pts(i,j,1) = pts(i,j,1) - cg\_pts(1);

pts(i,j,2) = pts(i,j,2) - cg\_pts(2);

end

end

% Generate the mean for all the sets of points and draw it

mean\_of\_sets = squeeze(mean(pts));

%Subtract the mean\_of\_sets from each of the point sets

for i = 1 : size(pts, 1)

for j = 1 : size(pts, 2)

pts(i,j,1) = pts(i,j,1) - mean\_of\_sets(j,1);

pts(i,j,2) = pts(i,j,2) - mean\_of\_sets(j,2);

end

end

% We have 21 sets of points and 68 points are present in each set ,

% in order to do principal component analysis, we need 21 x 136 (68x2)

% matrix, where, the matrix is as below:-

% 1 - [x1 y1 x2 y2 ..... x68 y68]

% 2 - [x1 y1 x2 y2 ..... x68 y68]

% ...............................

% 21- [x1 y1 x2 y2 ..... x68 y68]

mean\_sub\_set = zeros(size(pts,1), size(pts,2) \* size(pts,3));

for i = 1 : size(pts, 1)

for j = 1 : size(pts, 2)

% eg:- mean\_sub\_set(1,1) = pts(1,1,1) and mean\_sub\_set(1,2) = pts(1,1,2);

mean\_sub\_set(i,2\*j-1) = pts(i,j,1);

mean\_sub\_set(i,2\*j) = pts(i,j,2);

end

end

% Perform Principal Component Analysis on the above mean\_subtracted set

% COEFF - A p-by-p matrix, each column containing coefficients for one

% principal component. The columns are in order of decreasing component

% variance

% SCORE - the principal component scores

% latent - a vector containing the eigenvalues of the covariance matrix of X

[COEFF,SCORE,latent] = princomp(mean\_sub\_set);

% Extract the first 3 eigen values and eigen modes of variation

eigen\_values = latent(1:3);

eigen\_modes = COEFF(:,1:3);

meanPDM = mean\_of\_sets;

% Apply ASM for a series of Target Images found in testImg directory

files=dir('../testImg/\*.mhd');

for i = 1: length(files)

meanPDM = mean\_of\_sets;

fName = strcat('../testImg/',files(i).name);

targImg = loadMETA(fName);

% Compute the gradient of image and initialize parameters

[gradX, gradY] = gradient(targImg);

flag = true;

nIter = 1;

maxIter = 1000;

no\_of\_points = size(meanPDM,1);

finalPDM = zeros(no\_of\_points,2);

% Iterate until maxIter is reached or no change occurs in PDM

while flag

% Compute the normals

normals = lineNormals2D(meanPDM);

newPDM = zeros(no\_of\_points, 2);

% Scale back to the object space

scaledMeanPDM = cat(2,meanPDM(:,1) + mean\_x, meanPDM(:,2) + mean\_y);

for i = 1 : no\_of\_points

% Sample 13 points along the normal each 1 pixel apart

sample\_points = zeros(13,2);

for j = 1 : 13

sample\_points(j,:) = scaledMeanPDM(i,:) + (7-j) \* normals(i,:);

end

% Use bilinear interpolation to estimate gradients at sample

% points.

% Compute geometry to image match i.e,

% directional derivative = dot product of interpolated value and

% normals

interpol = bilinear(gradX, gradY, sample\_points);

dirDer = interpol \* normals(i,:)';

% Find the maximum directional derivative

[val ind] = max(abs(dirDer));

% Replace the point in scaledMeanPDM with that point which had

% the largest directional derivative (see computation of

% candidate point)

scaledMeanPDM(i,:) = scaledMeanPDM(i,:) + (7-ind) \* normals(i,:);

end

% Scale it back to mean object space

newPDM = cat(2,scaledMeanPDM(:,1) - mean\_x, scaledMeanPDM(:,2) - mean\_y);

diff = newPDM - mean\_of\_sets;

% eigen\_modes = (32x2) , i.e, each column vector is (x1 y1)..(xn yn)

% Taking the transpose, eige\_modes' becomes 2x32

% reshape re-arranges the diff' matrix to 32x1 matrix

% On multiplying above you get the beta matrix of 2x1

beta = eigen\_modes' \* reshape(diff',32,1);

% Restrict beta values to -3\*sigma and +3\*sigma

%(sigma =sqrt(eigen\_values)) , eigen\_values = variance

for i = 1 : 3

sigma = sqrt(eigen\_values(i));

if beta(i) > 3 \* sigma

beta(i) = 3 \* sigma;

elseif beta(i) < -3 \* sigma

beta(i) = -3 \* sigma;

end

end

% Convert shape to PDM (use original mean\_of\_sets and not meanPDM)

shapeSpacePDM = mean\_of\_sets + reshape(eigen\_modes\*beta,2,16)';

% Check if shapeSpacePDM == meanPDM with tolerance level of 0.0001

tolerance = 0.0001;

index = abs(shapeSpacePDM - meanPDM) <= tolerance;

hasNotMoved = all(all(index));

% If the newPDM didn't change i.e, no point moved, then exit the loop

% Also exit the loop if no. of iterations reached max.

if hasNotMoved || (nIter == maxIter)

flag = false;

%disp(nIter);

% Translate the result to the object space

finalPDM = cat(2,shapeSpacePDM(:,1) + mean\_x, shapeSpacePDM(:,2) + mean\_y);

else

meanPDM = shapeSpacePDM;

nIter = nIter + 1;

end

end

% Overlay the final result to the test MR image

overlayPDM2MR(finalPDM,fName,1);

end

lineNormals2D.m (used to compute normals)

function N=lineNormals2D(V,L)

% This function calculates the normals, of the line points

% using the neighbouring points of each contour point,

% forward and backward differences on the end points

%

% N=lineNormals2D(V,L)

%

% inputs,

% V : List of points/vertices M x 2

% (optional)

% Lines : A N x 2 list of line pieces, by indices of the vertices

% (if not set assume Lines=[1 2; 3 4 ; ... ; M-1 M])

%

% Concept Explanation

% ====================

% if we define dx=x2-x1 and dy=y2-y1, then the normals are (-dy, dx) and (dy, -dx).

% If no line-indices, assume a x(1) connected with x(2), x(3) with x(4) ...

if(nargin<2)

L=[(1:(size(V,1)-1))' (2:size(V,1))'];

end

% Calculate tangent vectors

DT=V(L(:,1),:)-V(L(:,2),:);

% Make influence of tangent vector 1/Distance

% (Weighted Central Differences. Points which are closer give a

% more accurate estimate of the normal)

LL=sqrt(DT(:,1).^2+DT(:,2).^2);

DT(:,1)=DT(:,1)./max(LL.^2,eps);

DT(:,2)=DT(:,2)./max(LL.^2,eps);

D1=zeros(size(V)); D1(L(:,1),:)=DT;

D2=zeros(size(V)); D2(L(:,2),:)=DT;

D=D1+D2;

% Normalize the normal

LL=sqrt(D(:,1).^2+D(:,2).^2);

N(:,1)=-D(:,2)./LL;

N(:,2)= D(:,1)./LL;

bilinear.m

%Bilinear Interpolation: function to perform bilinear interpolation

function [interpol] = bilinear(gradX, gradY, samplePoints)

interpol = zeros(size(samplePoints));

for i=1:size(samplePoints,1)

f = floor(samplePoints(i,:));

d = samplePoints(i,:) - f;

g00 = [gradX(f(1),f(2)), gradY(f(1),f(2))];

g01 = [gradX(f(1), f(2)+1), gradY(f(1), f(2))];

g10 = [gradX(f(1)+1,f(2)), gradY(f(1)+1,f(2))];

g11 = [gradX(f(1)+1, f(2)+1), gradY(f(1)+1, f(2)+1)];

i1 = g00\*(1-d(2)) + g01\*d(2);

i2 = g10\*(1-d(2)) + g11\*d(2);

interpol(i,:) = i1\*(1-d(1)) + i2\*d(1);

end

end

References

1. Snap tutorials
2. ‘An Introduction to Active Shape Models’, Tim Cootes
3. ‘Active Shape Models – their training and application’, Tim Cootes