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

**Ravikiran Janardhana**

**Medical Image Analysis – COMP 775**

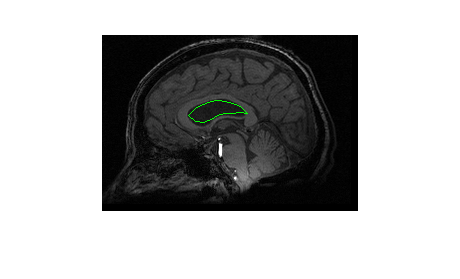
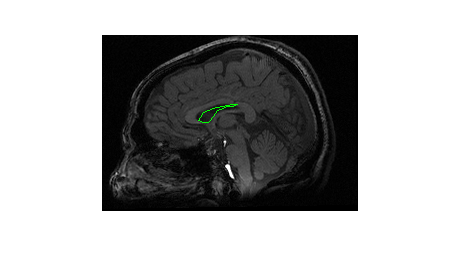
**Fall 2011**

**Tasks completed**

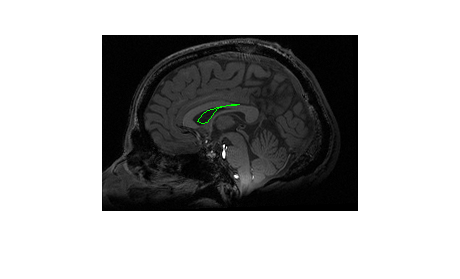
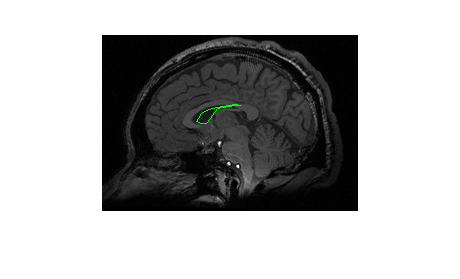
1. Wrote a Matlab program to carry out ASM (Active Shape Model) to find a dark region given a set of training Point Distribution Models and the target image.
2. Recorded the segmented point distribution model (output) for various target images and overlaid it on the original MRI slice image to view the result.

**Output**

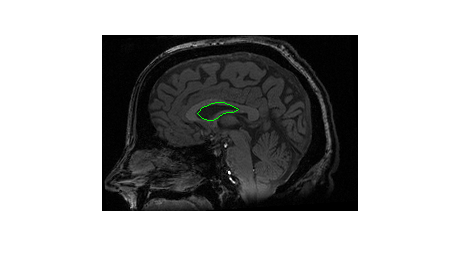
1) Input file - 30.2d.mhd 2) Input file - 39.2d.mhd

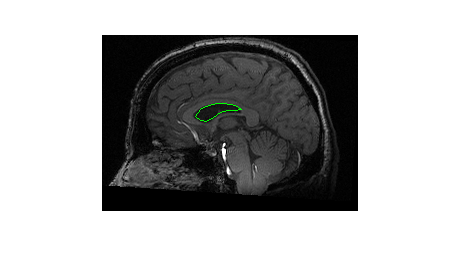
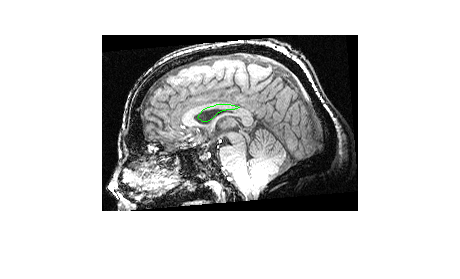
3) Input file - 45.2d.mhd 4) Input file - 55.2d.mhd

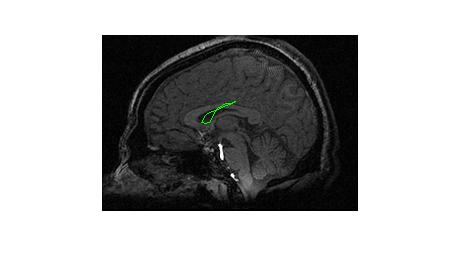
5) Input file - 58.2d.mhd 6) Input file - 68.2d.mhd

7) Input file - 81.2d.mhd 8) Input file - 89.2d.mhd

9) Input file - 93.2d.mhd 10) Input file – 34.2d.mhd

**ASM Implementation**

*Input: A set of training Point Distribution Models and target images.*

*Output: Segmented Point Distribution model*

A brief outline of the ASM implementation is as follows:-

1. Read the set of training point distribution models. Center each tuple by subtracting its center of gravity, computes the mean of the results (meanOrigPDM), and translationally align each centered tuple to that mean.
2. Compute the mean subtracted set and apply Principal Component Analysis using princomp function in Matlab.
3. Record the first 3 eigenmodes and the associated principal variances, i.e, eigenvalues.
4. Read the target image and compute the gradient in X and Y direction.
5. Begin Iterations
6. Compute the normal to the mean point distribution model (meanPDM).
7. Scale the meanPDM to the object space by reversing the translation applied before. Add [mean\_x mean\_y] to each tuple to obtain the scaledMeanPDM.
8. For each point in scaledMeanPDM, sample 13 points (6 on each side) along its normal.
9. Interpolate (bilinear interpolation) the gradient at these sample points using the 2D gradient of the target image.
10. Calculate the directional derivative (dot product of interpolated values and normals).
11. In the newPDM, move the point to the sample point which has the maximum directional derivative value.
12. Scale back the newPDM to meanobject space by applying translation.
13. Project the newPDM to the shape space of first 3 eigen modes and 3 eigen values as follows:

*betai = {eigenModesTi (newPDM – meanOrigPDM)}/eigenValuesi i=1,2,3*

*Restrict beta values to [-3 \* σ, 3 \* σ]*

1. Calculate candidate PDM in shape space:

shapeSpacePDM = meanOrigPDM + ∑(betai\*eigenmodes­i)

1. If shapeSpacePDM = meanPDM OR maximum number of iterations are reached, goto Step 11, else meanPDMnew = shapeSpacePDM and goto Step 1 of the iteration.
2. Return the final shapeSpacePDM at the end.

To be the on the safer side, the maximum number of iterations is set to 1000.

**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