Automatic MRI-T1W Brain Tissue Segmentation with Image Noise Filtering

Matthew Basso
Ryerson University

***Abstract*—Accurate segmentation of brain tissues from magnetic resonance imaging (MRI), is extremely important as there are many clinical and research applications. Developing an accurate segmentation algorithm that is robust to noise, bias filed, and partial volume effects while, using an unsupervised segmentation method is a challenging feat. This paper purposes to use different filtering methods and k-means clustering to segment the brain tissue into three classes; white matter (WM), grey matter (GM), and cerebrospinal fluid (CSF). Validation metrics were used to evaluate the performance of both the filtering methods and the segmentation. The results show that the wiener filter was the preferred filtering method, whereas k-means segmentation had DSC scores of 0.756, 0.694, and 0.710 respectfully for CSF, GM, and WM tissue classes.**

***Index Terms*—MRI, segmentation, k-means clustering, DSC**

I. Introduction

Medical images have become essential tools for physicians and healthcare professionals in the medical diagnosis and treatment of disease. To get a visual representation of a patients anatomy, different imaging modalities can be used to capture radiological images of soft tissue, bones, and vessels. Some of these modalities include: x-ray, medical resonance imaging (MRI), computer-assisted tomography (CT), and full-field digital mammography (FFDM).

The MRI is an important medical device that healthcare professionals use in the diagnosis and analysis of the brain. This modality has been widely used in both clinical applications and in neuroscience research, and is preferred over other modalities as it has the advantage of sharp soft-tissue contrast and high spatial resolution [1]. Due to these advantages, this modality can produce significant differentiation and classification between different tissue types including white matter (WM), grey matter (GM), and cerebrospinal fluid (CSF), which can be seen in figure 1 [1]. Currently, brain tissue segmentation in MR images play an important role in clinical research and has been a rapidly growing field of study. The classification of these tissues can be used for the planning and evaluation of drug therapy, disease detection, and measurement of brain size, shape and homogeneity [1]. Manual segmentation of brain tissues is not performed due to it being time consuming and prone to large intraobserver and interobserver variation [1]. Therefore, over the past two decades automatic approaches have been proposed for precise brain tissue segmentation [1].

Although deep learning algorithm's have been gaining much popularity due to their dominance in terms of accuracy when trained on large datasets, there become issues when they are trained on small datasets. Most medical imaging databases contain few ground truths that deep learning algorithms can be trained on. Therefore due to this, much research has been focused on developing unsupervised algorithms that do not need data to train, while still providing accurate results.

MR images are usually introduced with noise during the acquisition phase, which degrades the image quality and affects the accuracy in diagnosis of disease [2]. In general, MR images are prone to artifacts such as partial volume averaging, bias, tissue variation, and noise. The types of noise found in MR images are speckle noise, Gaussian noise and salt and pepper noise which ultimately influences the image quality [2]. Equation (1), illustrates the image formation model which is used to describe the corrupted image S which is formed by the interaction between the bias field B, bias-free image I, and the additive noise η [3].

$$S(x, y) = I(x, y)B(x, y) + \eta(x, y) \tag{1}$$

The quality of an image plays a vital role in the performance of image processing, feature extraction, classification, and segmentation algorithms. Specifically when performing segmentation tasks, noise can have a drastic affect isolating specific structures [2]. Therefore, to filter and remove the noise without losing important information about the image, pre-processing steps must be implemented. This paper proposes various filters which include the median, average, and wiener filters which are then compared using noise performance metrics to determine which filter bests removes the noise while still preserving the image information.

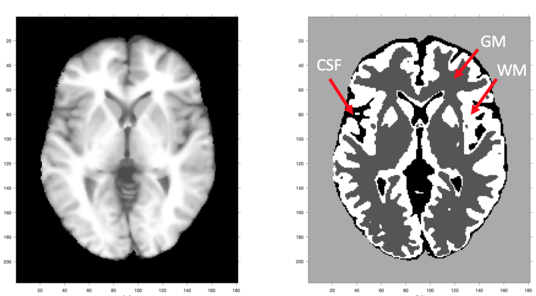

Fig. 1: Original brain image (left). Segmented brain tissues: CSF, GM, & WM (right).

II. Materials and Methods

A. Data

The database used in this paper is composed of neurological T1-weighted MR images with a variety of noise levels (1%, 3%, 5%, 7%, 9%) and artifacts. The database also contains ground truth segmentation's for the WM, GM, CSF tissue classes which are used to validate the performance of the algorithm.

B. Experimental Design

This paper discusses various pre-processing filtering methods used for noise removal and the segmentation of three different brain tissue classes. For both image enhancement and segmentation, validation metrics were gathered and discussed. Figure 2 shows a flowchart representation of the proposed system. In the first step, the MR images are loaded with the ground truths and brain mask's. In the second step, skull stripping is implemented which involves the multiplication of the brain mask with the loaded MR images. This removes the skull and isolates only the brain structure. Next, the median, averaging, and wiener filtering methods were proposed for the noise removal in the images. Validation metrics were then used to assess the performance of these filters individually. The best noise removal filter was then used in the next step for segmentation. In the fourth step, k-means segmentation was performed on the wiener filtered images. Four clusters were used to segment the image into background, CSF, GM, and WM. Dice Similarity Coefficient (DSC), Overlap Fraction (OF), and Extra Fraction (EF) validation metrics were then used to assess the performance of the segmentation method and these metrics were then compared to non-filtered volumes.

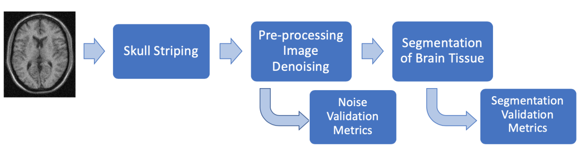

Fig. 2: Experimental design flowchart.

C. Pre-Processing

This paper analyzes and evaluates the performance of different filtering methods as a function of increasing noise level. The different filtering methods examined in this work include the average filter, median filter, and wiener filter. The filtering methods were evaluated by both qualitative and quantitative methods. For qualitative methods, the image quality is observed visually at the highest noise level. However, quantitative evaluation's were performed using statistical calculation's such as Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index (SSIM).

1) Average (Mean) Filter: The average (mean) filter is one of the most simplest spatial filters which operates by using a sliding window that replaces the center value in the window with the average of all the values in the window [2]. This filter is used to suppress noise in an image however, the main drawback of this filter is that it is poor in edge preserving causing blurring and smoothing of the edges [2]. The average filter is given by equation 2 [2].

$$f(x,y) = \frac{1}{mn} \sum_{(s,t)\epsilon S_{xy}} g(s,t) \tag{2}$$

where *g(s,t)* is the noisy image and S_{xy} is the sliding window area where the average is computed [2]. The *f(x,y)* represents the average filtered image computed using the pixels in region S_{xy} with a window size $m \times n$ [2].

2) Median Filter: The median filter is a non-linear filter that provides excellent noise reduction while overcoming the main limitations of the average filter, preserving edge content with less blurring [2]. Similar to the average filter, the median filter uses a sliding window and replaces the center value in the window by the statistical median of its $m \times n$ neighbourhood rather than its mean [2]. The median operator orders the values in the neighbourhood window at every pixel location increasing the computational expense which is one of its disadvantages [2]. However, a major advantage of the median filter is its ability to eliminate large magnitude impulse noise such as 'salt and pepper' noise while still preserving edge content [2]. This filtering method still however slightly blurs edge content in the presence of noise. The median filter is given by (3) where, X_i and Y_i are the noisy input and denoised output region at location i of the filter. The $[W_i]r$ is the order statistic of the samples inside the window W_i. In this experiment a 3×3 window size for the filter was used as [2] states that a larger window size will increase the MSE.

$$Y_i = med\{W_i\} = med\{X_i + r : r\epsilon W\} \tag{3}$$

3) Wiener Filter: The Wiener filter is an adaptive filtering method for deconvolution where an image is blurred by a low-pass filter and inverse filtering is performed to recover the image [2]. This inverse filtering however, is highly sensitive to additive noise therefore, an optimal trade-off between inverse filtering and noise smoothing is found [2]. The Wiener filter gives an optimal mean square error as it minimizes it in the process of inverse filtering and noise smoothing [2]. This filter is more precise than other linear filters as it preserves most edge content and high frequency components in the image. The downfall to this filter is that it is more computationally expensive than other linear filters [2]. The Wiener filter is given by (4) where, *H(m,n)* and $H*(m,n)$ is the degradation function and its complex conjugate. While $P_n(m,n)$ and $P_s(m,n)$ is the power spectral density of the noise and the un-degraded image [2].

$$W_{(m,n)} = \frac{H*(m,n)}{\mid H(m,n) \mid^2 + \frac{P_n(m,n)}{P_s(m,n)}} \tag{4}$$

D. Filtering Performance Metrics

The three types of filtering performance metrics used in evaluating the filters include, Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity

Index (SSIM). MSE calculates the average squared intensity of an estimated image and compares it to a reference image [4]. This metric measures the error between the estimated and reference image and outputs a value of zero if the images are alike and a non-negative value if the images are different [2]. PSNR looks at contrast adjustments in an image [4]. This metric measures the peak dynamic range of an image after reconstruction [2]. A high value for PSNR indicates a good reconstruction however, a low PSNR means a bad reconstruction and is expressed in units of dB. The last filtering performance metric used was SSIM which, measures the similarity between the structures of two images [4]. Both the MSE ans PSNR approaches measure absolute error while, SSIM is a perception-based model that takes into account image structural degradation [4]. This structural information is computed using three terms, the luminance term, contrast term, and structural term. The SSIM metric is a multiplicative combination of the three terms [4].

E. Segmentation

1) k-means clustering: Over the years, medical image segmentation has been one of the most important parts of medical image processing. Image segmentation, is a procedure whereby region of interest's (ROI's) are extracted from images through either an automatic or semi-automatic process [5]. The segmentation algorithm outputs an image of labels referring to the ROI's extracted. In medical image analysis, tissue segmentation plays a vital role as it can be used in different applications such as for the analysis of anatomical structures, and disease diagnosis [1]. The fundamental tissue structures of the brain include: white matter (WM), gray matter (GM), and cerebrospinal fluid (CSF). If these tissues are segmented from a MR brain volume, analysis can be performed on the individual tissue classes and metrics such as size, shape, and texture can be extracted and be correlated to disease [1].

In this paper, k-means clustering was performed to segment the brain into different tissue classes. K-means clustering is a widely used unsupervised classification algorithm which separates data into k classes based on the randomization of initial centroids of each class [5]. Since MR images are 2-dimensional, all pixels are reshaped to form a 1-dimensional vector. For an image, the initial cluster is formed by associating each pixel in the image to the given nearest centroid using a distance measure, then the mean values of the elements in the clusters are computed and the centroids are replaced by them [5]. This process is done iteratively until there is no change in cluster centers [5]. One of the disadvantages of this algorithm is the number of clusters must be known. In this paper, four clusters were chosen, one for the black background and three for the tissue types. Another disadvantage is that this algorithm is sensitive to noise therefore, pre-processing techniques are performed prior to the segmentation. The k-means algorithm is given below:

Segmentation Algorithm: K-Means clustering
Step 1: The 2-D image is reshaped to form a 1-D vector.
Step 2: The number of clusters k, and the initial cluster centers c, are chosen.
Step 3: Each pixel is allocated to the nearest class by minimizing the objective function J which can be seen in equation (5). Where, $\|x_i^j - c_j\|^2$ is the distance measure while x_i^j is the pixel value and c_j is the cluster center [5].

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \|x_i^j - c_j\|^2 \tag{5}$$

Step 4: The mean is computed for each cluster and the cluster center moves to this mean value.
Step 5: Steps 3 and 4 are repeated until no more new centroids are created. The image is then reshaped back to 2-D.

F. Segmentation Performance Metrics

To quantitatively measure the performance of the algorithms automatic segmentation three metrics were used that compare the automated results to ground truth data which was manually segmented by an expert. After performing k-means clustering, a labeled image is created and each label corresponds to one of the three tissue types on a per slice basis. Based on the classification results, the following statistical parameters were computed by totaling the results from all pixels in the image: number of true positives (TP), number of false positives (FP), number of true negatives (TN), and number of false negatives (FN). From these statistical parameters the Dice Similarity Coefficient (DSC), Overlap Fraction (OF), and Extra Fraction (EF) were calculated for each segmented class.

1) Dice Similarity Coefficient (DSC): The DSC is a metric used to measure the effectiveness of the intersection between a segmented object and the ground truth [6]. Once the labels are found corresponding to the tissue types, each tissue types DSC is found by using the TP, FP, and FN statistical parameters which can be seen in equation (6). The average DSC is then taken for each tissue slice in the entire volume.

$$DSC = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{6}$$

2) Overlap Fraction (OF): The OF measures the tissue area that is correctly classified, with respect to the ground truth area [6]. This metric is similar to the DSC metric as intersection between objects is taken into account. The OF is found using the TP and FN statistical parameters which can be seen in equation (7).

$$OF = \frac{TP}{TP + FN} \tag{7}$$

3) Extra Fraction (EF): The third metric, known as EF measures the area that is falsely classified as a specific segmented tissue, relative to the ground truth tissue area [6]. The EF is found using the TP, FP, and FN statistical parameters which can be seen in equaiton (8).

$$EF = \frac{FP}{TP + FN} \tag{8}$$

III. Results

In this section, two analyses are performed, one being filtering method validation and the other being segmentation method validation.

A. Filter Validation Metrics

1) Qualitative Analysis: Figure 3 presents MR images at a slice thickness of 9mm and a noise level of 9% which is the worst possible additive noise found in the database. This figure shows the non-filtered original image, the median filtered image, the averaging filtered image, and lastly the wiener filtered image. From visual inspection it can be observed that the wiener filter performed the best when removing the Gaussian noise while still preserving the structural information of the brain. The median and averaging filter however, caused blurring of the edge content in the images. The visual interpretation is supported by quantitative measurements.

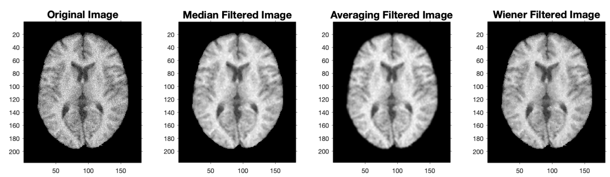

Fig. 3: MR images at a slice thickness of 9mm and a noise level of 9% with different filtering methods used.

2) Quantitative Analysis: Figure 4 illustrates the average MSE, PSNR, and SSIM across all slice thicknesses for each noise level. The noise present in the images were Gaussian that ranged from: 1%, 3%, 5%, 7%, 9% in intensity. When observing the MSE plot, it can be seen that the wiener filter had the lowest squared intensity difference between the filtered and reference image while, both the median and averaging filter had more error. Since the wiener filter had the lowest MSE across all noise levels, this means that most of the impulse noise was removed from the image while still preserving the structural information at the lower noise levels.

The second plot in the figure shows the PSNR in dB for all three filters. PSNR measures the peak dynamic range of an image after reconstruction. A high PSNR value indicates that there is good reconstruction of the filtered image in relation to the reference image. Again when inspecting the plot, the wiener filter had a higher PSNR value across all noise levels rather than the median and averaging filter.

The last plot in the figure shows the SSIM for all three filters. It can be seen that from noise levels 1-7, the wiener filter has very similar structural information as the reference image however at noise level 9, the wiener filter performs the worst and the averaging filter performs the best. This is due to the blurring which is produced by the averaging filter which at a high noise levels maintains the similar structures as the reference image. Overall, the wiener filter performed the best across all filtering validation metrics and through qualitative analysis. Therefore, this filter was selected as the filter to use when removing noise from the images which would then be used for segmentation.

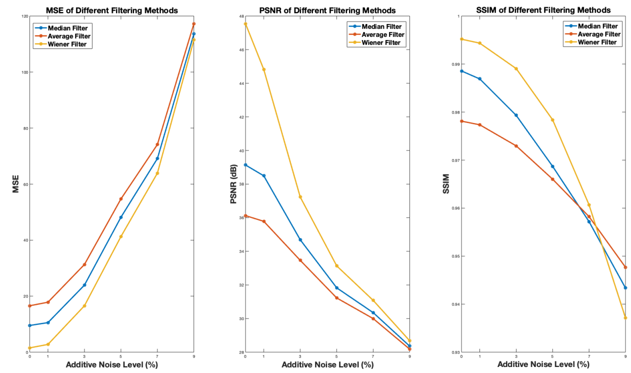

Fig. 4: MSE plot of filtering methods (left). PSNR plot of filtering methods (middle). SSIM plot of filtering methods (right).

B. Segmentation Validation Metrics

Figure 5 illustrates the brain tissue segmentation results of the image volume with a slice thickness of 7 and a noise level of 5%. This figure shows the ground truths provided, the segmented tissue classes, and an overlap of the ground truth and segmented tissue. This overlap is a visual representation of how the DSC, OF, and EF metrics are calculated. In this slice, both the ground truths and segmented tissue classes look very similar in structure which means that the k-means segmentation performed well.

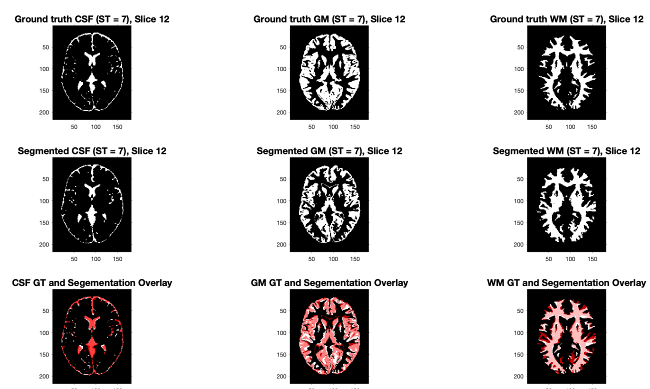

Fig. 5: Ground truths of tissue classes (top). Segmented tissue classes (middle). Overlay of segmented tissues over ground truths (bottom).

The comparison between the un-filtered vs wiener filtered image DSC's can be seen in figure 6, where the first plot shows the DSC in relation to increasing noise levels. From this plot the DSC for the filtered images outperforms the non-filtered images as noise level increases. Another observation is that, as noise level increases, the segmentation performance for the non-filtered images drastically decreases. This makes sense since there is more noise, it is harder for the k-means algorithm to differentiate between different tissue types. When looking at noise level 9 for the filtered images, the DSC's are not as low and there is only a slight drop in performance. This ultimately means that the filtering method helped the segmentation method to perform well.

The second plot in the figure shows the DSC's of the filtered tissue classes vs slice thickness. From observation it can be seen that as slice thickness increases, DSC decreases. This makes sense since as slice thickness increases the tissue intensities are averaged in the 2-D slice. This type of noise is called partial volume averaging and causes blurring in the images reducing the edge information between tissues. Therefore, for the best segmentation accuracy, a smaller slice thickness is preferred to eliminate partial volume averaging.

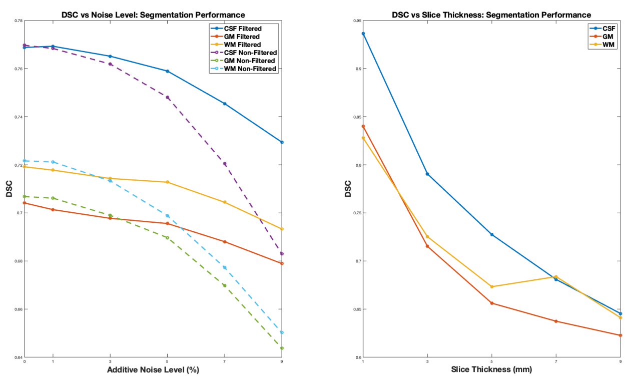

Fig. 6: DSC vs noise level (left). DSC vs slice thickness (right).

The OF and EF segmentation validation metrics were used to quantitatively validate the segmentation performance of the k-means algorithm. The OF measures the tissue area that is correctly classified, with respect to the ground truth area while the EF measures the area that is falsely classified, relative to the ground truth tissue area. From table I it can be seen that for the OF metric, there was an increase in correctly classified area for all tissue classes after filtering was performed. Also it was interesting to notice that the CSF, OF metric was higher than GM tissue. This was an interesting result since there is a more abundant amount of GM than CSF in the brain. When looking at the EF metric, WM tissue was the highest which meant that there was a a large amount of falsely classified tissue, relative to the ground truth tissue area.

TABLE I: Overlap Factor and Extra Fraction Metrics.

	OF CSF	OF GM	OF WM
Filtered	0.756	0.694	0.710
Non-Filtered	0.742	0.689	0.697
	EF CSF	EF GM	EF WM
Filtered	0.767	0.648	0.880
Non-Filtered	0.762	0.641	0.860

IV. Discussion & Conclusion

In this paper, a complete workflow for brain tissue segmentation was designed and developed. The T1W MR images used in this paper ranged in percentage of noise level and slice thickness. The three different filtering methods used to remove partial volume averaging, bias, tissue variation, and noise were; median filter, averaging filter, and wiener filter. To validate which filter performed the best in terms of noise reduction and tissue structure preservation, three metrics were used; MSE, PSNR, and SSIM. These were used to compare the filtered images to the reference images. From these results it was observed that the wiener filter outperformed all other filtering methods even as noise level increased. Images were filtered using the wiener filter and k-means segmentation was performed to segment the brain tissues into four tissue classes; background, CSF, GM, and WM. This unsupervised method was used due to its robustness against additive noise in images. Lastly, to validate the performance of the segmentation algorithm, statistical performance metrics were found and used in the DSC, OF, and EF metrics. These metrics were analyzed as a function of noise level and partial volume averaging. It was observed that as noise level increased DSC decreased for all tissue classes. Also it was viewed that DSC decreased rapidly as slice thickness increased, which was due to the partial volume averaging. Overall, the proposed algorithm outperformed the non-filtered images and resulted in DSC values of 0.756, 0.694, and 0.710 respectfully for CSF, GM, and WM tissue classes.

In future work, an investigation of other filtering methods and segmentation methods will be analyzed. Specifically, the wavelet and bilateral filtering will be looked into as they are edge-preserving filtering methods. Also, more tradition segmentation methods such as Otsu's method and Gaussian mixture models will be applied and compared with the proposed k-means approach. Finally, the designed algorithm will be integrated into a complete CAD tool for radiologists to use.

References

[1] Y. Kong, X. Chen, J. Wu, P. Zhang, Y. Chen, and H. Shu, "Automatic brain tissue segmentation based on graph filter," *BMC medical imaging*, vol. 18, no. 1, p. 9, 2018.

[2] I. S. Isa, S. N. Sulaiman, M. Mustapha, and S. Darus, "Evaluating denoising performances of fundamental filters for t2-weighted mri images," *Procedia Computer Science*, vol. 60, pp. 760–768, 2015.

[3] S. Song, Y. Zheng, and Y. He, "A review of methods for bias correction in medical images," *Biomedical Engineering Review*, vol. 1, no. 1, 2017.

[4] S. Saladi and N. Amutha Prabha, "Analysis of denoising filters on mri brain images," *International Journal of Imaging Systems and Technology*, vol. 27, no. 3, pp. 201–208, 2017.

[5] T. Kalaiselvi, N. Kalaichelvi, and P. Sriramakrishnan, "Automatic brain tissues segmentation based on self initializing k-means clustering technique," *International Journal of Intelligent Systems and Applications*, vol. 9, no. 11, p. 52, 2017.

[6] A. Khademi, "Medical image processing techniques for the objective quantification of pathology in magnetic resonance images of the brain," Ph.D. dissertation, 2012.

Appendix:

```
% Copyright: Matthew Basso, 2020
% Not to be distributed or modified.

% Assignment 1 -  Unsupervised Segmentation in MRI

clear
close all
clc
warning('off');

%% Filtering and Segmentation Algorithm

tic;

ST = [1,3,5,7,9];
noise = [0,1,3,5,7,9];
method = ["k-means"]; % Segmentation Methods :"k-means","otsu","gaussian_mixture"

path_MR = '/Users/matthewbasso/Desktop/Assignment_1_path/MRI';
path_GT = '/Users/matthewbasso/Desktop/Assignment_1_path/groundtruth';

fprintf('Segementation Method: %s \n',method);

vol_noise_performance_ST = zeros(length(noise),12,length(ST));
vol_seg_performance_ST = zeros(length(noise),21,length(ST));
vol_seg_performance_ST_no_filtering = zeros(length(noise),21,length(ST));

for st = 1:length(ST)

  for n = 1:length(noise)

    disp(['Slice Thickenss: ',num2str(ST(st)),', Additive Noise Level: ', num2str(noise(n)),'%']);

    % Load in Brain Images with GT's

    [vol,vol_no_noise,csf, gm, wm] = load_brain_images(ST(st),noise(n),path_MR,path_GT);

    % Pre-processing (Median filter, averaging filter, and weiner filter)

    win_size = 3;
```

```
    [vol_med_filt,vol_average_filt,vol_weiner_filt,Noise_performance] =
preprocessing_filtering(vol,vol_no_noise,win_size,'false');

    % Segmentation and Performance

    attempts = 3;
    clusters = 4;
    I_filt = vol_weiner_filt;
    I_non_filt = vol;

    [Seg_performance,Seg_performance_no_filtering] =
segmentation_and_performance(I_filt,I_non_filt,ST(st),clusters,method,attempts,csf,gm,wm,'false');


    % Storing all stats

    vol_noise_performance_ST(n,:,st) = mean(Noise_performance(any(Noise_performance ~= 0,2),:),1);

    Seg_performance(any(isnan(Seg_performance), 2), :) = [];
    vol_seg_performance_ST(n,:,st) = mean(Seg_performance,1);

    Seg_performance_no_filtering(any(isnan(Seg_performance_no_filtering), 2), :) = [];
    vol_seg_performance_ST_no_filtering(n,:,st) = mean(Seg_performance_no_filtering,1);

  end

end

toc;

GTscroll
%% Noise Performance Plots

noise_performance_stats = noise_performance_plots(vol_noise_performance_ST,noise);

%% Segmentation Performance Plots

[segmentation_performance_stats,non_filtered_segmentation_performance_stats] =
segmentation_performance_plots(vol_seg_performance_ST,vol_seg_performance_ST_no_filtering,noise,
ST);


function [average_filt] = avg_filter(I,win_size)
```

```
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

h = fspecial('average',win_size);

average_filt = imfilter(I, h);


end

function [score, precission, recall] = bfscore(TP,FP,FN)
%UNTITLED10 Summary of this function goes here
%   Detailed explanation goes here


%% Precission

precission = TP/(TP+FP);

%% Recall

recall = TP/(TP+FN);

%% Score

score = (2*precission*recall)/(recall+precission);

end


function [TP,TN,FP,FN] = confusion_matrix(gt,predicted)
%UNTITLED7 Summary of this function goes here
%   Detailed explanation goes here

%% Finding TP, TN, FP, FN

if isa(gt,'double')
    gt = imbinarize(gt);
end

if isa(predicted,'double')
    predicted = imbinarize(predicted);
end
```

```
TP = length(find(predicted ==1 & gt == 1));
TN = length(find(predicted ==0 & gt == 0));
FP = length(find(predicted ==1 & gt == 0));
FN = length(find(predicted ==0 & gt == 1));


end

function [dsc] = DSC(TP,FP,FN)
%UNTITLED8 Summary of this function goes here
%   Detailed explanation goes here


dsc = (2*TP)/(2*TP + FP + FN);

end


function [ef] = EF(FP,TP,FN)
%UNTITLED9 Summary of this function goes here
%   Detailed explanation goes here

%% Extra Fraction Measure

ef = FP/(TP+FN);

end


clear
clc

ST= 3;
noise = 7;

% load T1 MRI with ST = 3, and noise level = 7;
[vol] = load_brain(ST,noise);

% show the slice 15
figure; imshow(vol(:,:,15), []);title(['Original MRI Image, Slice 15'])
```

```
% load corresponding ground truth masks
[csf, gm, wm, brainMask] = load_brain_GT(ST);

% remove skull (and keep only brain region)
vol = vol .* brainMask;

% view images
figure; imshow(vol(:,:,15), []); title(['Brain Extracted MRI Image (ST=3), Slice 15'])
figure; imshow(csf(:,:,15), []); title(['Ground truth CSF (ST=3), Slice 15'])
figure; imshow(wm(:,:,15), []); title(['Ground truth WM (ST=3), Slice 15'])
figure; imshow(gm(:,:,15), []); title(['Ground truth GM (ST=3), Slice 15'])


function GTscroll(fig)
% GTscroll  Use the scroll wheel to navigate figure windows
%
% function GTscroll(fig)
%
%    Updated the figure WindowScrollWheelFcn so that
%    the mouse scroll wheel can be used to cycle
%    through the figures.
%
%    One can set the default figure create function to
%    automatically activate this function when a new
%    figure is created with the following command,
%      GTscroll('install');
%    and it can be removed from the DefaultFigureCreateFcn
%    with,
%      GTscroll('uninstall');
%
% Example - To apply this to all figures
%   GTscroll;
%
% Example - To apply this to only one figure
%   GTscroll(gcf);
%
% Gus - Jan 2008

  if (nargin>0 && ischar(fig)),
    % Remove function from DefaultFigureCreateFcn
    set(0,'DefaultFigureCreateFcn',regexprep(get(0,'DefaultFigureCreateFcn'),'GTscroll\(gcf\);',''));
    switch lower(fig),
      case ('install'),  % add function to DefaultFigureCreateFcn
```

```
      disp('Setting DefaultFigureCreateFcn');
      set(0,'DefaultFigureCreateFcn',['GTscroll(gcf); ' get(0,'DefaultFigureCreateFcn')]);
    case ('uninstall'),  % Already removed
      return
  end;
  CH = get(0,'children');
 elseif nargin>0,
  CH = fig;
 else
  CH = get(0,'children');
 end;

 % cycle through figures
 for ii = 1:length(CH),
  % the zoom function casues problems and should be deactivated first
  Hz = zoom(CH(ii));
  if strcmpi(Hz.enable,'on'),
    Hz.enable = 'off';
    zm = 1;  % remember to reactivate zoom
  else
    zm = 0;
  end;
  % set WindowScrollWheelFcn, but dsplay warning if replacing existing
  % function
  if ~isempty(get(CH(ii),'WindowScrollWheelFcn')),
    warning('GTscroll:info','GTscroll has replaced the original WindowScrollWheelFcn on Figure
%g',CH(ii));
  end;
  set(CH(ii),'WindowScrollWheelFcn',@figScroll);
  % reactivate zoom
  if (zm), Hz.enable = 'on'; end;
 end;

%% -------------------------------
%   function figScroll(src,evnt)
%     % get figure handles and sort them
%     H = sort(get(0,'children'));
%     if (evnt.VerticalScrollCount>0),
%       % scroll down
%       F = find(H>src,1);
%     else
%       % scroll up
%       F = find(H<src,1,'last');
```

```
%     end;
%     if isempty(F),
%       % jump to first or last figure
%       if (evnt.VerticalScrollCount<0),
%         figure(H(end));
%       else
%         figure(H(1));
%       end;
%     else
%       % goto next figure
%       figure(H(mod(F-1,length(H))+1));
%     end;
%   end

function figScroll(src,evnt)
    % get figure handles and sort them
    H = sort(get(0,'children'));
    [~, in]=sort([H.Number]);
    H=H(in);
    if (evnt.VerticalScrollCount>0),
        % scroll down
        %F = find(H>src,1);
        if isempty(find([H.Number]<src.Number,1)),
            F=max([H.Number]);
        else
            F=find([H.Number]<src.Number,1,'last'); %valerio
        end
    else
        % scroll up
        %F = find(H<src,1,'last');
        if isempty(find([H.Number]>src.Number,1)),
            F=min([H.Number]);
        else
            F=find([H.Number]>src.Number,1); %valerio
        end
    end;
    if isempty(F),
        % jump to first or last figure
        if (evnt.VerticalScrollCount<0),
            figure(H(end));
        else
            figure(H(1));
        end;
```

```
    else
        % goto next figure
        figure(H(mod(F-1,length(H))+1));
    end;
end

end

function [noise_performance_stats] = noise_performance_plots(vol_noise_performance_ST,noise)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

% Noise Performance Plots

n = length(noise);

noise_performance_stats(1:n) =
struct('Noise_Level',[],'MSE_No_Filt',[],'MSE_Median',[],'MSE_Avg',[],...

'MSE_Wiener',[],'PSNR_No_Filt',[],'PSNR_Median',[],'PSNR_Avg',[],'PSNR_Wiener',[],'SSIM_No_Filt',
[],'SSIM_Median',[],'SSIM_Avg',[],...
    'SSIM_Wiener',[],'Total_vol_stats',[]);

vol_noise_performance = mean(vol_noise_performance_ST,3);

for i = 1:n

    noise_performance_stats(i).Noise_Level = noise(i);
    noise_performance_stats(i).MSE_No_Filt = vol_noise_performance(i,10);
    noise_performance_stats(i).MSE_Median  = vol_noise_performance(i,1);
    noise_performance_stats(i).MSE_Avg  = vol_noise_performance(i,2);
    noise_performance_stats(i).MSE_Wiener  = vol_noise_performance(i,3);
    noise_performance_stats(i).PSNR_No_Filt  = vol_noise_performance(i,11);
    noise_performance_stats(i).PSNR_Median  = vol_noise_performance(i,4);
    noise_performance_stats(i).PSNR_Avg  = vol_noise_performance(i,5);
    noise_performance_stats(i).PSNR_Wiener  = vol_noise_performance(i,6);
    noise_performance_stats(i).SSIM_No_Filt  = vol_noise_performance(i,12);
    noise_performance_stats(i).SSIM_Median  = vol_noise_performance(i,7);
    noise_performance_stats(i).SSIM_Avg  = vol_noise_performance(i,8);
    noise_performance_stats(i).SSIM_Wiener  = vol_noise_performance(i,9);
    noise_performance_stats(i).Total_vol_stats = vol_noise_performance_ST;

end
```

```
%% MSE

figure('units','normalized','outerposition',[0 0 1 1])
subplot(1,3,1)
plot(noise,vol_noise_performance(:,1),'-o','LineWidth',3);
hold on
plot(noise,vol_noise_performance(:,2),'-o','LineWidth',3);
plot(noise,vol_noise_performance(:,3),'-o','LineWidth',3);

hold off
xticks(noise);
title('MSE of Different Filtering Methods','fontweight','bold','fontsize',20);
ylabel('MSE','fontweight','bold','fontsize',20);
xlabel('Additive Noise Level (%)','fontweight','bold','fontsize',20);
legend('Median Filter','Average Filter','Wiener Filter','fontweight','bold','fontsize',16);

%% PSNR

subplot(1,3,2)
plot(noise,vol_noise_performance(:,4),'-o','LineWidth',3);
hold on
plot(noise,vol_noise_performance(:,5),'-o','LineWidth',3);
plot(noise,vol_noise_performance(:,6),'-o','LineWidth',3);

hold off
xticks(noise);
title('PSNR of Different Filtering Methods','fontweight','bold','fontsize',20);
ylabel('PSNR (dB)','fontweight','bold','fontsize',20);
xlabel('Additive Noise Level (%)','fontweight','bold','fontsize',20);
legend('Median Filter','Average Filter','Wiener Filter','fontweight','bold','fontsize',16);

%% Structural Similarity Index (SSIM) for measuring image quality

subplot(1,3,3)
plot(noise,vol_noise_performance(:,7),'-o','LineWidth',3);
hold on
plot(noise,vol_noise_performance(:,8),'-o','LineWidth',3);
plot(noise,vol_noise_performance(:,9),'-o','LineWidth',3);

hold off
xticks(noise);
title('SSIM of Different Filtering Methods','fontweight','bold','fontsize',20);
```

```
ylabel('SSIM','fontweight','bold','fontsize',20);
xlabel('Additive Noise Level (%)','fontweight','bold','fontsize',20);
legend('Median Filter','Average Filter','Wiener Filter','fontweight','bold','fontsize',16);

%% Scroll

GTscroll;
end


function [hist_norm] = histogram_norm(I,gHIR, gLIR)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here


hist_norm = (I - min(I(:))).*((gHIR - gLIR)./(max(I(:))- min(I(:)))+ gLIR);

end

function [int_norm] = intensity_scaling(I,gHIR, gLIR)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

%% Intensity Scaling

int_norm = (I - gLIR)/(gHIR - gLIR);

end


% Copyright: April Khademi, 2012
% Not to be distributed or modified.
function [vol] = load_brain(ST, noise,path)

currentFolder = path;
filename = ['/t1_icbm_normal_',num2str(ST)','mm_pn',num2str(noise)','_rf0.rawb'];

x = 181;
y = 217;

if(ST == 1)
   z = 181;
elseif(ST == 3)
```

```
    z = 60;
elseif(ST == 5)
    z = 36;
elseif(ST == 7)
    z = 26;
elseif(ST == 9)
    z = 20;
end

% read in the volume
% Read in the data specified by fid (and typecast to double)
% reshape into images
% fid = fopen([currentFolder, '/MRI/volumes/', filename], 'r');
fid = fopen([currentFolder, filename], 'r');

v = double(fread(fid));
vol = reshape(v, x,y,z);

% Rotate so in axial direction
vol = imrotate(vol, 90);

fclose all;

end

% Copyright: April Khademi, 2012
% Not to be distributed or modified.

% Load brainweb ground truth data
% get binary images for ground truth of CSF, WM and GM tissues
function [csf, gm, wm, brainMask] = load_brain_GT(ST,path)

currentFolder = path;

% maskDir     = [currentFolder, '/groundtruth/groundtruth/'];
maskDir     = [currentFolder];

crispFile   = ['/phantom_',num2str(ST)',' .0mm_normal_crisp.hdr'];

x = 181;
y = 217;
% Specify the number of slices (as a function of slice thickness
if(ST == 1)
```

```
    z = 181;
elseif(ST == 3)
    z = 60;
elseif(ST == 5)
    z = 36;
elseif(ST == 7)
    z = 26;
elseif(ST == 9)
    z = 20;
end

if(ST ~= 1)
% read in the volume
% Read in the data specified by fid (and typecast to double)
% reshape into images

phantom = analyze75read([maskDir, crispFile]);
% phantom = imrotate(phantom, 180);

else

    fid = fopen([maskDir, crispFile(1:end-3),'raw']);
    % Read in the data specified by fid (and typecast to double)
    phantom = double(fread(fid));
    % reshape into images
    phantom = reshape(phantom, x,y,z);
    phantom = imrotate(phantom, 90);

end

csf = zeros(size(phantom));
wm = zeros(size(phantom));
gm = zeros(size(phantom));
brainMask = zeros(size(phantom));

% Get discrete classifications
% 0=Background, 1=CSF, 2=Grey Matter, 3=White Matter, 4=Fat, 5=Muscle/Skin,
% 6=Skin, 7=Skull, 8=Glial Matter, 9=Connective, 10 =MS lesion

% Count CSF and Glial matter as one
ind = find(phantom == 1);
csf(ind) = 1;
brainMask(ind) = 1;
```

```
% Find GM
ind = find(phantom == 2);
gm(ind) = 2;
brainMask(ind) = 1;

% Find WM
ind = find(phantom == 3);
wm(ind) = 3;
brainMask(ind) = 1;
ind = find(phantom == 8);
wm(ind) = 3;
brainMask(ind) = 1;

fclose all;

end


function [vol,vol_no_noise,csf, gm, wm] = load_brain_images(ST,noise,path_MR,path_GT)
%This function loads in brain images and outputs brain volumes
%according to noise level and slice thickness


%% Loading in T1 MRI Volume with ST, and noise level

[vol] = load_brain(ST,noise,path_MR);

%% Loading in T1 with no noise

[vol_no_noise] = load_brain(ST,0,path_MR);

%% Loading in Groundtruth T1 MRI Volume with ST, and noise level

[csf, gm, wm, brainMask] = load_brain_GT(ST,path_GT);

%% Remove skull (and keep only brain region) for volume with noise and no noise

vol = vol .* brainMask;

vol_no_noise = vol_no_noise.*brainMask;
```

```
end


function [Mean_Square_Error] = mse(I,I_filt)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here

[M,N] = size(I);

Mean_Square_Error = (norm(double(I_filt(:)) - double(I(:)),2).^2) / (M * N);

end


function [of] = overlap_factor(TP,FN)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

%% Overlap Factor Measure

of = TP/(TP+FN);

end


% OverlayImgs
% Show an image with a transparent mask superimposed.
%
% function overlayImgs(img, binaryOverlay, overlayColor)
% img              - rows x cols x bands image
% binaryOverlay    - rows x cols binary image
% overlayColor     - length 3 color vector, each component between 0 and 1

function overlay(img, binaryOverlay, overlayColor)

if(size(img,3) == 3)

img(:,:,1) = (img(:,:,1) - min(min(img(:,:,1)))) / (max(max(img(:,:,1))) - min(min(img(:,:,1))));
img(:,:,2) = (img(:,:,2) - min(min(img(:,:,2)))) / (max(max(img(:,:,2))) - min(min(img(:,:,2))));
img(:,:,3) = (img(:,:,3) - min(min(img(:,:,3)))) / (max(max(img(:,:,3))) - min(min(img(:,:,3))));

colorImg = cat(3, overlayColor(1) * binaryOverlay, ...
   overlayColor(2) * binaryOverlay, overlayColor(3) * binaryOverlay);
```

```
binaryOverlay = repmat(binaryOverlay, [1 1 3]);
if size(img, 3) == 1
    img = repmat(img, [1 1 3]);
end
% imshow(uint8(img + 0.5 * binaryOverlay .* (colorImg - img)));
imshow((img + 0.5 * binaryOverlay .* (colorImg - img)));

else
  img(:,:,1) = (img(:,:,1) - min(min(img(:,:,1)))) / (max(max(img(:,:,1))) - min(min(img(:,:,1))));

colorImg = cat(3, overlayColor(1) * binaryOverlay, ...
    overlayColor(2) * binaryOverlay, overlayColor(3) * binaryOverlay);

binaryOverlay = repmat(binaryOverlay, [1 1 3]);
if size(img, 3) == 1
    img = repmat(img, [1 1 3]);
end
% imshow(uint8(img + 0.5 * binaryOverlay .* (colorImg - img)));
imshow((img + 0.5 * binaryOverlay .* (colorImg - img)));
end


function [results] = performance(csf, gm, wm, imlabel,vol,method,ST,slice,show)

% Performance


%% Label

if strcmp(method,"k-means")

    if sum(imlabel(:)) == size(imlabel,1)*size(imlabel,2)

        seg_csf = zeros(size(imlabel));
        seg_gm = zeros(size(imlabel));
        seg_wm = zeros(size(imlabel));

    else

        cluster1 = vol.*uint8((imlabel == 1));
        cluster2 = vol.*uint8((imlabel == 2));
        cluster3 = vol.*uint8((imlabel == 3));
```

```
        cluster4 = vol.*uint8((imlabel == 4));

        mean_clust = [mean(nonzeros(cluster1),'all'), mean(nonzeros(cluster2),'all'),...
            mean(nonzeros(cluster3),'all'),mean(nonzeros(cluster4),'all')];

        mean_clust(isnan(mean_clust))=0;

        [~,I] = sort(mean_clust,2);
        background = imlabel == I(1);
        seg_csf = imlabel == I(2);
        seg_gm = imlabel == I(3);
        seg_wm = imlabel == I(4);

    end

elseif strcmp(method,"otsu")

    if sum(imlabel(:)) == size(imlabel,1)*size(imlabel,2)

        seg_csf = zeros(size(imlabel));
        seg_gm = zeros(size(imlabel));
        seg_wm = zeros(size(imlabel));

    else

        cluster1 = vol.*uint8(imlabel == 1);
        cluster2 = vol.*uint8(imlabel == 2);
        cluster3 = vol.*uint8(imlabel == 3);
        cluster4 = vol.*uint8(imlabel == 4);

        mean_clust = [mean(nonzeros(cluster1),'all'), mean(nonzeros(cluster2),'all'),...
            mean(nonzeros(cluster3),'all'),mean(nonzeros(cluster4),'all')];

        mean_clust(isnan(mean_clust))=0;

        [~,I] = sort(mean_clust,2);
        background = imlabel == I(1);
        seg_csf = imlabel == I(2);
        seg_gm = imlabel == I(3);
        seg_wm = imlabel == I(4);

    end
```

```
elseif strcmp(method,"gaussian_mixture")

    cluster1 = vol.*uint8(imlabel == 1);
    cluster2 = vol.*uint8(imlabel == 2);
    cluster3 = vol.*uint8(imlabel == 3);

    mean_clust = [mean(nonzeros(cluster1),'all'), mean(nonzeros(cluster2),'all'),...
        mean(nonzeros(cluster3),'all')];

    mean_clust(isnan(mean_clust))=0;

    if sum(sum(csf+gm+wm)) == 0

        seg_csf = zeros([size(csf,1) size(csf,2)]);
        seg_gm = zeros([size(csf,1) size(csf,2)]);
        seg_wm = zeros([size(csf,1) size(csf,2)]);

    else

        if and(mean_clust(1) > mean_clust(2),mean_clust(1) > mean_clust(3))

            seg_wm = imbinarize(cluster1);

        elseif and(mean_clust(1) < mean_clust(2),mean_clust(1) < mean_clust(3))

            seg_csf = imbinarize(cluster1);

        else

            seg_gm = imbinarize(cluster1);

        end


        if and(mean_clust(2) > mean_clust(1),mean_clust(2) > mean_clust(3))

            seg_wm = imbinarize(cluster2);

        elseif and(mean_clust(2) < mean_clust(1),mean_clust(2) < mean_clust(3))

            seg_csf = imbinarize(cluster2);

        else
```

```
            seg_gm = imbinarize(cluster2);

        end


        if and(mean_clust(3) > mean_clust(2),mean_clust(3) > mean_clust(1))

            seg_wm = imbinarize(cluster3);

        elseif and(mean_clust(3) < mean_clust(2),mean_clust(3) < mean_clust(1))

            seg_csf = imbinarize(cluster3);

        else

            seg_gm = imbinarize(cluster3);

        end

    end

else

    error('Error: Incorrect Method Name!');

end

%% Plot show GT vs Segmented
if strcmp(show,'true')

    figure('units','normalized','outerposition',[0 0 1 1]);
    subplot(3,3,1)
    imshow(csf);
    title(['Ground truth CSF (ST = ',num2str(ST),'), Slice ',num2str(slice)],'fontweight','bold','fontsize',16);
    subplot(3,3,2)
    imshow(gm);
    title(['Ground truth GM (ST = ',num2str(ST),'), Slice ',num2str(slice)],'fontweight','bold','fontsize',16);
    subplot(3,3,3)
    imshow(wm);
    title(['Ground truth WM (ST = ',num2str(ST),'), Slice ',num2str(slice)],'fontweight','bold','fontsize',16);
    subplot(3,3,4)
    imshow(seg_csf);
```

```
    title(['Segmented CSF (ST = ',num2str(ST),'), Slice ',num2str(slice)],'fontweight','bold','fontsize',16);
    subplot(3,3,5)
    imshow(seg_gm);
    title(['Segmented GM (ST = ',num2str(ST),'), Slice ',num2str(slice)],'fontweight','bold','fontsize',16);
    subplot(3,3,6)
    imshow(seg_wm);
    title(['Segmented WM (ST = ',num2str(ST),'), Slice ',num2str(slice)],'fontweight','bold','fontsize',16);

    subplot(3,3,7)
    csf_overlay = labeloverlay(csf,seg_csf,'Colormap','autumn','Transparency',0.25);
    imshow(csf_overlay);
    title('CSF GT and Segementation Overlay','fontweight','bold','fontsize',16)
    subplot(3,3,8)
    gm_overlay = labeloverlay(gm,seg_gm,'Colormap','autumn','Transparency',0.25);
    imshow(gm_overlay);
    title('GM GT and Segementation Overlay','fontweight','bold','fontsize',16)
    subplot(3,3,9)
    wm_overlay = labeloverlay(wm,seg_wm,'Colormap','autumn','Transparency',0.25);
    imshow(wm_overlay);
    title('WM GT and Segementation Overlay','fontweight','bold','fontsize',16)

end


%% Classification Results

[TP_csf,~,FP_csf,FN_csf] = confusion_matrix(csf,seg_csf);
[TP_gm,~,FP_gm,FN_gm] = confusion_matrix(gm,seg_gm);
[TP_wm,~,FP_wm,FN_wm] = confusion_matrix(wm,seg_wm);

%% Dice Similarity Coefficient

[dsc_csf] = DSC(TP_csf,FP_csf,FN_csf);
[dsc_gm] = DSC(TP_gm,FP_gm,FN_gm);
[dsc_wm] = DSC(TP_wm,FP_wm,FN_wm);

%% Precision, Recall, & F1-score

[precision_csf,recall_csf,f1_csf] = precession_recall_f1(TP_csf,FN_csf,FP_csf);
[precision_gm,recall_gm,f1_gm] = precession_recall_f1(TP_gm,FN_gm,FP_gm);
[precision_wm,recall_wm,f1_wm] = precession_recall_f1(TP_wm,FN_wm,FP_wm);

%% Extra Fraction
```

```
[ef_csf] = EF(FP_csf,TP_csf,FN_csf);
[ef_gm] = EF(FP_gm,TP_gm,FN_gm);
[ef_wm] = EF(FP_wm,TP_wm,FN_wm);

%% Overlap Factor Measure

[of_csf] = overlap_factor(TP_csf,FN_csf);
[of_gm] = overlap_factor(TP_gm,FN_gm);
[of_wm] = overlap_factor(TP_wm,FN_wm);

%% VDR Metric

[vdr_csf] = VDR(FP_csf,FN_csf,TP_csf);
[vdr_gm] = VDR(FP_gm,FN_gm,TP_gm);
[vdr_wm] = VDR(FP_wm,FN_wm,TP_wm);

%% Classification Result Vector

results = [dsc_csf,dsc_gm,dsc_wm,of_csf,of_gm,of_wm,...
    precision_csf,precision_gm,precision_wm,recall_csf,recall_gm,recall_wm,...
    f1_csf,f1_gm,f1_wm,ef_csf,ef_gm,ef_wm,vdr_csf,vdr_gm,vdr_wm];


end


function [precision,recall,f1] = precession_recall_f1(TP,FN,FP)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here

%% Precision
precision = TP / (TP + FP);

%% Recall
recall = TP / (TP + FN);

%% F1-score
f1 = 2 * (precision * recall) / (precision + recall);

end
```

```
function [med_filt,average_filt,weiner_filt, filter_performance] =
preprocessing(I,I_no_noise,win_size,show)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here

%% Filters

% Median filter
med_filt = medfilt2(I,[win_size win_size]);

% Averaging filter
average_filt = avg_filter(I,win_size);

% Weiner filter
weiner_filt = wiener2(I,[win_size win_size]);

%% Mean Squared Error (MSE)

[MSE_med] = mse(med_filt,I_no_noise);
[MSE_avg] = mse(average_filt,I_no_noise);
[MSE_weiner] = mse(weiner_filt,I_no_noise);
[MSE_no_filt] = mse(I,I_no_noise);

%% Signal-to-noise ratio

[psnr_median, ~] = psnr_snr(med_filt, I_no_noise);
[psnr_avg, ~] = psnr_snr(average_filt, I_no_noise);
[psnr_weiner, ~] = psnr_snr(weiner_filt, I_no_noise);
[psnr_no_filt, ~] = psnr_snr(I, I_no_noise);

%% Structural Similarity Index (SSIM) for measuring image quality

[ssimval_median,~] = ssim(med_filt, I_no_noise);
[ssimval_avg,~] = ssim(average_filt, I_no_noise);
[ssimval_weiner,~] = ssim(weiner_filt, I_no_noise);
[ssimval_no_filt,~] = ssim(I, I_no_noise);

%% Filters Performance Measurement

filter_performance =
[MSE_med,MSE_avg,MSE_weiner,psnr_median,psnr_avg,psnr_weiner,ssimval_median,ssimval_avg,ssi
mval_weiner,...
   MSE_no_filt,psnr_no_filt,ssimval_no_filt];
```

```
%% Figures of Filtered Images
if strcmp(show,'true')

    figure;
    subplot(1,4,1)
    imshow(I,[]);
    title('Original Image','fontweight','bold','fontsize',16);

    subplot(1,4,2)
    imshow(med_filt,[]);
    title('Median Filtered Image','fontweight','bold','fontsize',16);

    subplot(1,4,3)
    imshow(average_filt,[]);
    title('Averaging Filtered Image','fontweight','bold','fontsize',16);

    subplot(1,4,4)
    imshow(weiner_filt,[]);
    title('Wiener Filtered Image','fontweight','bold','fontsize',16);

end

end


function [vol_med_filt,vol_average_filt,vol_weiner_filt,Noise_performance] =
preprocessing_filtering(vol,vol_no_noise,win_size,plot_show)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here


Noise_performance = zeros(size(vol,3),12,1);
vol_med_filt = zeros(size(vol,1,2,3));
vol_average_filt = zeros(size(vol,1,2,3));
vol_weiner_filt = zeros(size(vol,1,2,3));

for i = 1:size(vol,3)

    I = uint8(vol(:,:,i));
    I_no_noise = uint8(vol_no_noise(:,:,i));

    if sum(sum(I))>0
```

```
        [med_filt,average_filt,weiner_filt,filter_performance] =
preprocessing(I,I_no_noise,win_size,plot_show);

    else

        filter_performance = zeros(1,12);
        med_filt = zeros(size(I));
        average_filt = zeros(size(I));
        weiner_filt = zeros(size(I));

    end

    Noise_performance(i,:,1) = filter_performance;

    vol_med_filt(:,:,i) = med_filt;
    vol_average_filt(:,:,i) = average_filt;
    vol_weiner_filt(:,:,i) = weiner_filt;
end


end


function [peaksnr,snr] = psnr_snr(I,ref)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

%% MSE
err = mse(I,ref);

%% PSNR
peakval = diff(getrangefromclass(I));
peaksnr = 10*log10(peakval.^2/err);

%% SNR

if isinteger(ref)
    ref = double(ref);
end

snr = 10*log10(mean(ref(:).^2)/err);
```

```
end


function [vol_imlabel] = segmentation(vol,clusters,method,attempts,show)
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here

% k-Means

if strcmp(method,"k-means")

    for i = 1:size(vol,3)

        [slice_imlabel,~] = imsegkmeans(vol(:,:,i),clusters,'NumAttempts',attempts);

        if strcmp(show,'true')

            figure
            subplot(1,2,1);
            imshow(vol(:,:,i),[]);
            xlabel('(a)','fontweight','bold','fontsize',16);

            subplot(1,2,2);
            imshow(slice_imlabel,[]);
            xlabel('(b)','fontweight','bold','fontsize',16);

        end

        vol_imlabel(:,:,i) = slice_imlabel;

    end

    GTscroll;

    % Otsu thresholding

elseif strcmp(method,"otsu")

    for i = 1:size(vol,3)
        thresh = multithresh(vol(:,:,i),clusters-1);
        imlabel = imquantize(vol(:,:,i),thresh);
```

```
        vol_imlabel(:,:,i) = imlabel;

        if strcmp(show,'true')

            figure
            ax1 = subplot(1,3,1);
            imagesc(vol(:,:,i));
            colormap(ax1,gray);
            title('Original Image')

            ax2 = subplot(1,3,2);
            imagesc(imlabel);
            colormap(ax2,gray);
            colorbar;
            title('Otsu Labeled Image');

            subplot(1,3,3);
            histogram(imlabel);
            title('Histogram of Labeled Image');
            xlabel('Label');
            ylabel('Number of Occurences');

        end
    end

    GTscroll;

elseif strcmp(method,"gaussian_mixture")

    for i = 1:size(vol,3)

        vol_img = vol(:,:,i);

        if sum(sum(vol_img))>0

            Igm = single(vol_img(vol_img>0));

            options = statset('Display','off','MaxIter',1000,'TolFun',1e-5);
            gm = fitgmdist(Igm,clusters-1,'RegularizationValue',0.01,'Options',options);

            idx = cluster(gm,single(vol_img(:)));

            imlabel = reshape(idx,[size(vol_img,1) size(vol_img,2)]);
```

```
        else

            imlabel = ones(size(vol_img,1),size(vol_img,2));

        end

        vol_imlabel(:,:,i) = imlabel;

        if strcmp(show,'true')

            figure
            ax1 = subplot(1,3,1);
            imagesc(vol_img);
            colormap(ax1,gray);
            title('Original Image')

            ax2 = subplot(1,3,2);
            imagesc(imlabel);
            colormap(ax2,gray);
            colorbar;
            title('Gaussian Mixture Model Labeled Image');

            subplot(1,3,3);
            histogram(imlabel);
            title('Histogram of Labeled Image');
            xlabel('Label');
            ylabel('Number of Occurences');
        end
    end

    GTscroll;

else
    error('Error: Incorrect method name!');

end

end


function [Seg_performance,Seg_performance_no_filtering] =
segmentation_and_performance(I_filt,I_non_filt,ST,clusters,method,attempts,csf,gm,wm,plot_show)
```

```
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here

I_filt = uint8(I_filt);
I_non_filt = uint8(I_non_filt);

%% Segmentation with Preprocessing

[imlabel] = segmentation(I_filt,clusters,method,attempts,plot_show);

%% Segmentation with No Preprocessing

[imlabel_no_filtering] = segmentation(I_non_filt,clusters,method,attempts,plot_show);

%% Segmentation Performance

Seg_performance = zeros(size(I_filt,3),21);
Seg_performance_no_filtering = zeros(size(I_filt,3),21);

for i = 1:size(I_filt,3)

    gt_csf = csf(:,:,i);
    gt_gm = gm(:,:,i);
    gt_wm = wm(:,:,i);

    Seg_performance(i,:) = performance(gt_csf, gt_gm, gt_wm, imlabel(:,:,i), I_filt(:,:,i), method,
ST,i,plot_show);
    Seg_performance_no_filtering(i,:) = performance(gt_csf, gt_gm, gt_wm, imlabel_no_filtering(:,:,i),
I_non_filt(:,:,i), method, ST,i,plot_show);

end


GTscroll;
end


function [segmentation_performance_stats,non_filtered_segmentation_performance_stats] =
segmentation_performance_plots(vol_seg_performance_ST,vol_seg_performance_ST_no_filtering,noise,
ST)
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here
```

```
% Filtered Segmentation Performance Plots

n = length(noise);
st = length(ST);

segmentation_performance_stats(1:n) = struct('Noise_Level',[],'DSC_CSF',[],'DSC_GM',[],...
    'DSC_WM',[],'OF_CSF',[],'OF_GM',[],'OF_WM',[],'Precision_CSF',[],'Precision_GM',[],...
    'Precision_WM',[],'Recall_CSF',[],'Recall_GM',[],'Recall_WM',[],...
    'f1_CSF',[],'f1_GM',[],'f1_WM',[],'EF_CSF',[],'EF_GM',[],'EF_WM',[],...
    'VDR_CSF',[],'VDR_GM',[],'VDR_WM',[],'Total_vol_stats',[]);

vol_seg_performance = mean(vol_seg_performance_ST,3);
vol_seg_performance_no_filtering = mean(vol_seg_performance_ST_no_filtering,3);

for i = 1:n

    segmentation_performance_stats(i).Noise_Level = noise(i);
    segmentation_performance_stats(i).DSC_CSF  = vol_seg_performance(i,1);
    segmentation_performance_stats(i).DSC_GM  = vol_seg_performance(i,2);
    segmentation_performance_stats(i).DSC_WM  = vol_seg_performance(i,3);
    segmentation_performance_stats(i).OF_CSF  = vol_seg_performance(i,4);
    segmentation_performance_stats(i).OF_GM  = vol_seg_performance(i,5);
    segmentation_performance_stats(i).OF_WM  = vol_seg_performance(i,6);
    segmentation_performance_stats(i).Precision_CSF  = vol_seg_performance(i,7);
    segmentation_performance_stats(i).Precision_GM  = vol_seg_performance(i,8);
    segmentation_performance_stats(i).Precision_WM  = vol_seg_performance(i,9);
    segmentation_performance_stats(i).Recall_CSF  = vol_seg_performance(i,10);
    segmentation_performance_stats(i).Recall_GM  = vol_seg_performance(i,11);
    segmentation_performance_stats(i).Recall_WM  = vol_seg_performance(i,12);
    segmentation_performance_stats(i).f1_CSF  = vol_seg_performance(i,13);
    segmentation_performance_stats(i).f1_GM  = vol_seg_performance(i,14);
    segmentation_performance_stats(i).f1_WM  = vol_seg_performance(i,15);
    segmentation_performance_stats(i).EF_CSF  = vol_seg_performance(i,16);
    segmentation_performance_stats(i).EF_GM  = vol_seg_performance(i,17);
    segmentation_performance_stats(i).EF_WM  = vol_seg_performance(i,18);
    segmentation_performance_stats(i).VDR_CSF  = vol_seg_performance(i,19);
    segmentation_performance_stats(i).VDR_GM  = vol_seg_performance(i,20);
    segmentation_performance_stats(i).VDR_WM  = vol_seg_performance(i,21);
    segmentation_performance_stats(i).Total_vol_stats = vol_seg_performance_ST;

end

% Non-Filtered Segmentation Performance Plots
```

```
non_filtered_segmentation_performance_stats(1:n) =
struct('Noise_Level',[],'DSC_CSF',[],'DSC_GM',[],...
    'DSC_WM',[],'OF_CSF',[],'OF_GM',[],'OF_WM',[],'Precision_CSF',[],'Precision_GM',[],...
    'Precision_WM',[],'Recall_CSF',[],'Recall_GM',[],'Recall_WM',[],...
    'f1_CSF',[],'f1_GM',[],'f1_WM',[],'EF_CSF',[],'EF_GM',[],'EF_WM',[],...
    'VDR_CSF',[],'VDR_GM',[],'VDR_WM',[],'Total_vol_stats',[]);

for i = 1:n

    non_filtered_segmentation_performance_stats(i).Noise_Level = noise(i);
    non_filtered_segmentation_performance_stats(i).DSC_CSF  = vol_seg_performance_no_filtering(i,1);
    non_filtered_segmentation_performance_stats(i).DSC_GM  = vol_seg_performance_no_filtering(i,2);
    non_filtered_segmentation_performance_stats(i).DSC_WM  = vol_seg_performance_no_filtering(i,3);
    non_filtered_segmentation_performance_stats(i).OF_CSF  = vol_seg_performance_no_filtering(i,4);
    non_filtered_segmentation_performance_stats(i).OF_GM  = vol_seg_performance_no_filtering(i,5);
    non_filtered_segmentation_performance_stats(i).OF_WM  = vol_seg_performance_no_filtering(i,6);
    non_filtered_segmentation_performance_stats(i).Precision_CSF  =
vol_seg_performance_no_filtering(i,7);
    non_filtered_segmentation_performance_stats(i).Precision_GM  =
vol_seg_performance_no_filtering(i,8);
    non_filtered_segmentation_performance_stats(i).Precision_WM  =
vol_seg_performance_no_filtering(i,9);
    non_filtered_segmentation_performance_stats(i).Recall_CSF  =
vol_seg_performance_no_filtering(i,10);
    non_filtered_segmentation_performance_stats(i).Recall_GM  =
vol_seg_performance_no_filtering(i,11);
    non_filtered_segmentation_performance_stats(i).Recall_WM  =
vol_seg_performance_no_filtering(i,12);
    non_filtered_segmentation_performance_stats(i).f1_CSF  = vol_seg_performance_no_filtering(i,13);
    non_filtered_segmentation_performance_stats(i).f1_GM  = vol_seg_performance_no_filtering(i,14);
    non_filtered_segmentation_performance_stats(i).f1_WM  = vol_seg_performance_no_filtering(i,15);
    non_filtered_segmentation_performance_stats(i).EF_CSF  = vol_seg_performance_no_filtering(i,16);
    non_filtered_segmentation_performance_stats(i).EF_GM  = vol_seg_performance_no_filtering(i,17);
    non_filtered_segmentation_performance_stats(i).EF_WM  = vol_seg_performance_no_filtering(i,18);
    non_filtered_segmentation_performance_stats(i).VDR_CSF  =
vol_seg_performance_no_filtering(i,19);
    non_filtered_segmentation_performance_stats(i).VDR_GM  =
vol_seg_performance_no_filtering(i,20);
    non_filtered_segmentation_performance_stats(i).VDR_WM  =
vol_seg_performance_no_filtering(i,21);
    non_filtered_segmentation_performance_stats(i).Total_vol_stats =
vol_seg_performance_ST_no_filtering;
```

```
end

%% DSC vs Additive Noise

figure('units','normalized','outerposition',[0 0 1 1]);

plot(noise,vol_seg_performance(:,1),'-o','LineWidth',3);
hold on
plot(noise,vol_seg_performance(:,2),'-o','LineWidth',3);
plot(noise,vol_seg_performance(:,3),'-o','LineWidth',3);
hold off
xticks(noise);

title('DSC vs Noise Level: Segmentation Performance','fontweight','bold','fontsize',16);
ylabel('DSC','fontweight','bold','fontsize',16);
xlabel('Additive Noise Level (%)','fontweight','bold','fontsize',16);
legend('CSF','GM','WM','fontweight','bold','fontsize',12);

%% DSC of Filtered vs Non-Filtered Images

figure('units','normalized','outerposition',[0 0 1 1]);
subplot(1,2,1)

plot(noise,vol_seg_performance(:,1),'-o','LineWidth',3);
hold on
plot(noise,vol_seg_performance(:,2),'-o','LineWidth',3);
plot(noise,vol_seg_performance(:,3),'-o','LineWidth',3);

plot(noise,vol_seg_performance_no_filtering(:,1),'--o','LineWidth',3);
plot(noise,vol_seg_performance_no_filtering(:,2),'--o','LineWidth',3);
plot(noise,vol_seg_performance_no_filtering(:,3),'--o','LineWidth',3);
hold off
xticks(noise);

title('DSC vs Noise Level: Segmentation Performance','fontweight','bold','fontsize',20);
ylabel('DSC','fontweight','bold','fontsize',20);
xlabel('Additive Noise Level (%)','fontweight','bold','fontsize',20);
legend('CSF Filtered','GM Filtered','WM Filtered','CSF Non-Filtered','GM Non-Filtered','WM
Non-Filtered','fontweight','bold','fontsize',16);

%% DSC vs ST
```

```
subplot(1,2,2)

plot(ST,reshape(mean(vol_seg_performance_ST(:,1,:),1),1,st),'-o','LineWidth',3);
hold on
plot(ST,reshape(mean(vol_seg_performance_ST(:,2,:),1),1,st),'-o','LineWidth',3);
plot(ST,reshape(mean(vol_seg_performance_ST(:,3,:),1),1,st),'-o','LineWidth',3);
hold off
xticks(ST);

title('DSC vs Slice Thickness: Segmentation Performance','fontweight','bold','fontsize',20);
ylabel('DSC','fontweight','bold','fontsize',20);
xlabel('Slice Thickness (mm)','fontweight','bold','fontsize',20);
legend('CSF','GM','WM','fontweight','bold','fontsize',16);

%% Overlap Fraction

figure('units','normalized','outerposition',[0 0 1 1]);
subplot(1,2,1)
plot(noise,vol_seg_performance(:,4),'-o','LineWidth',3);
hold on
plot(noise,vol_seg_performance(:,5),'-o','LineWidth',3);
plot(noise,vol_seg_performance(:,6),'-o','LineWidth',3);
hold off
xticks(noise);

title('OF vs Noise Level: Segmentation Performance','fontweight','bold','fontsize',20);
ylabel('OF','fontweight','bold','fontsize',20);
xlabel('Additive Noise Level (%)','fontweight','bold','fontsize',20);
legend('CSF','GM','WM','fontweight','bold','fontsize',16);

%% Extra Fraction

subplot(1,2,2)
plot(noise,vol_seg_performance(:,16),'-o','LineWidth',3);
hold on
plot(noise,vol_seg_performance(:,17),'-o','LineWidth',3);
plot(noise,vol_seg_performance(:,18),'-o','LineWidth',3);
hold off
xticks(noise);

title('EF vs Noise Level: Segmentation Performance','fontweight','bold','fontsize',20);
ylabel('EF','fontweight','bold','fontsize',20);
xlabel('Additive Noise Level (%)','fontweight','bold','fontsize',20);
```

```
legend('CSF','GM','WM','fontweight','bold','fontsize',16);

%% DSC before and after filtering

% Filtered Segementation
n0_CSF = reshape(vol_seg_performance_ST(1,1,:),1,st);
n1_CSF = reshape(vol_seg_performance_ST(2,1,:),1,st);
n3_CSF = reshape(vol_seg_performance_ST(3,1,:),1,st);
n5_CSF = reshape(vol_seg_performance_ST(4,1,:),1,st);
n7_CSF = reshape(vol_seg_performance_ST(5,1,:),1,st);
n9_CSF = reshape(vol_seg_performance_ST(6,1,:),1,st);

n0_GM = reshape(vol_seg_performance_ST(1,2,:),1,st);
n1_GM = reshape(vol_seg_performance_ST(2,2,:),1,st);
n3_GM = reshape(vol_seg_performance_ST(3,2,:),1,st);
n5_GM = reshape(vol_seg_performance_ST(4,2,:),1,st);
n7_GM = reshape(vol_seg_performance_ST(5,2,:),1,st);
n9_GM = reshape(vol_seg_performance_ST(6,2,:),1,st);

n0_WM = reshape(vol_seg_performance_ST(1,3,:),1,st);
n1_WM = reshape(vol_seg_performance_ST(2,3,:),1,st);
n3_WM = reshape(vol_seg_performance_ST(3,3,:),1,st);
n5_WM = reshape(vol_seg_performance_ST(4,3,:),1,st);
n7_WM = reshape(vol_seg_performance_ST(5,3,:),1,st);
n9_WM = reshape(vol_seg_performance_ST(6,3,:),1,st);

% Non-filtered Segementation

n0_CSF_no_filt = reshape(vol_seg_performance_ST_no_filtering(1,1,:),1,st);
n1_CSF_no_filt = reshape(vol_seg_performance_ST_no_filtering(2,1,:),1,st);
n3_CSF_no_filt = reshape(vol_seg_performance_ST_no_filtering(3,1,:),1,st);
n5_CSF_no_filt = reshape(vol_seg_performance_ST_no_filtering(4,1,:),1,st);
n7_CSF_no_filt = reshape(vol_seg_performance_ST_no_filtering(5,1,:),1,st);
n9_CSF_no_filt = reshape(vol_seg_performance_ST_no_filtering(6,1,:),1,st);

n0_GM_no_filt = reshape(vol_seg_performance_ST_no_filtering(1,2,:),1,st);
n1_GM_no_filt = reshape(vol_seg_performance_ST_no_filtering(2,2,:),1,st);
n3_GM_no_filt = reshape(vol_seg_performance_ST_no_filtering(3,2,:),1,st);
n5_GM_no_filt = reshape(vol_seg_performance_ST_no_filtering(4,2,:),1,st);
n7_GM_no_filt = reshape(vol_seg_performance_ST_no_filtering(5,2,:),1,st);
n9_GM_no_filt = reshape(vol_seg_performance_ST_no_filtering(6,2,:),1,st);

n0_WM_no_filt = reshape(vol_seg_performance_ST_no_filtering(1,3,:),1,st);
```

```
n1_WM_no_filt = reshape(vol_seg_performance_ST_no_filtering(2,3,:),1,st);
n3_WM_no_filt = reshape(vol_seg_performance_ST_no_filtering(3,3,:),1,st);
n5_WM_no_filt = reshape(vol_seg_performance_ST_no_filtering(4,3,:),1,st);
n7_WM_no_filt = reshape(vol_seg_performance_ST_no_filtering(5,3,:),1,st);
n9_WM_no_filt = reshape(vol_seg_performance_ST_no_filtering(6,3,:),1,st);

% DSC for CSF Before & After Filtering
figure('units','normalized','outerposition',[0 0 1 1]);

x=[n0_CSF n0_CSF_no_filt n1_CSF n1_CSF_no_filt n3_CSF n3_CSF_no_filt n5_CSF n5_CSF_no_filt
n7_CSF n7_CSF_no_filt n9_CSF n9_CSF_no_filt];
n=length(n0_CSF) ; xx=([1:12])'; % example
r=repmat(xx,1,n)';
g=r(:)';


positions = [1 2 3 4 5 6 7 8 9 10 11 12];
h=boxplot(x,g, 'positions', positions);
set(h,'linewidth',2)

set(gca,'xtick',[mean(positions(1:2)) mean(positions(3:4)) mean(positions(5:6)) mean(positions(7:8))
mean(positions(9:10)) mean(positions(11:12))])
set(gca,'xticklabel',{'0','1','3','5','7','9'},'Fontsize',12)

color = ['c', 'y', 'c', 'y','c', 'y','c', 'y','c', 'y','c', 'y'];
h = findobj(gca,'Tag','Box');

for j=1:length(h)
patch(get(h(j),'XData'),get(h(j),'YData'),color(j),'FaceAlpha',.5);
end
legend('Non-Filtered CSF','Filtered CSF');
xlabel('Additive Noise Level (%)','fontweight','bold','fontsize',16);
ylabel('DSC','fontweight','bold','fontsize',16);
title('DSC for CSF Before & After Filtering','fontweight','bold','fontsize',16);

% DSC for GM Before & After Filtering
figure('units','normalized','outerposition',[0 0 1 1]);

x=[n0_GM n0_GM_no_filt n1_GM n1_GM_no_filt n3_GM n3_GM_no_filt n5_GM n5_GM_no_filt
n7_GM n7_GM_no_filt n9_GM n9_GM_no_filt];
n=length(n0_GM) ; xx=([1:12])'; % example
r=repmat(xx,1,n)';
g=r(:)';
```

```
positions = [1 2 3 4 5 6 7 8 9 10 11 12];
h=boxplot(x,g, 'positions', positions);
set(h,'linewidth',2)

set(gca,'xtick',[mean(positions(1:2)) mean(positions(3:4)) mean(positions(5:6)) mean(positions(7:8))
mean(positions(9:10)) mean(positions(11:12))])
set(gca,'xticklabel',{'0','1','3','5','7','9'},'Fontsize',12)

color = ['c', 'y', 'c', 'y','c', 'y','c', 'y','c', 'y','c', 'y'];
h = findobj(gca,'Tag','Box');

for j=1:length(h)
patch(get(h(j),'XData'),get(h(j),'YData'),color(j),'FaceAlpha',.5);
end
legend('Non-Filtered CSF','Filtered CSF');
xlabel('Additive Noise Level (%)','fontweight','bold','fontsize',16);
ylabel('DSC','fontweight','bold','fontsize',16);
title('DSC for GM Before & After Filtering','fontweight','bold','fontsize',16);

% DSC for WM Before & After Filtering
figure('units','normalized','outerposition',[0 0 1 1]);

x=[n0_WM n0_WM_no_filt n1_WM n1_WM_no_filt n3_WM n3_WM_no_filt n5_WM n5_WM_no_filt
n7_WM n7_WM_no_filt n9_WM n9_WM_no_filt];
n=length(n0_WM) ; xx=([1:12])'; % example
r=repmat(xx,1,n)';
g=r(:)';


positions = [1 2 3 4 5 6 7 8 9 10 11 12];
h=boxplot(x,g, 'positions', positions);
set(h,'linewidth',2)

set(gca,'xtick',[mean(positions(1:2)) mean(positions(3:4)) mean(positions(5:6)) mean(positions(7:8))
mean(positions(9:10)) mean(positions(11:12))])
set(gca,'xticklabel',{'0','1','3','5','7','9'},'Fontsize',12)

color = ['c', 'y', 'c', 'y','c', 'y','c', 'y','c', 'y','c', 'y'];
h = findobj(gca,'Tag','Box');

for j=1:length(h)
```

```
patch(get(h(j),'XData'),get(h(j),'YData'),color(j),'FaceAlpha',.5);
end
legend('Non-Filtered CSF','Filtered CSF');
xlabel('Additive Noise Level (%)','fontweight','bold','fontsize',16);
ylabel('DSC','fontweight','bold','fontsize',16);
title('DSC for WM Before & After Filtering','fontweight','bold','fontsize',16);

%% Scroll

GTscroll;

end


function [svd] = SVD(dsc)
%UNTITLED4 Summary of this function goes here
%   Symmetric Volume Difference (SVD)
%       Provides a symmetric measure of the difference in volume of
%       the segmentation result and the reference shape.
%       Segmentation errors are estimated with SVD

svd = 1 - dsc;


end


function [vdr] = VDR(FP,FN,TP)
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here

vdr = abs(FP-FN)/(TP+FN);

end
```