



# **Lab 4:**

## **Atlas based segmentation (integration to the EM algorithm)**

Medical Image Segmentation and Applications

### **Team members:**

Juan Cisneros

Karla Sam

Stela Lila

## INTRODUCTION AND PROBLEM DEFINITION

When compared to other methods for image segmentation, the atlas-based segmentation has an ability to segment the image with no well-defined relation between regions and pixels' intensities. Another important advantage of atlases is in their use in clinical practice, for computer aided diagnosis whereas they are often used to measure the shape of an object or detect morphological differences between patient groups.

The use of an anatomic or probabilistic atlas is crucial in many medical image segmentation challenges to provide spatially-consistent segmentation results. The implementation of a probabilistic atlas will be incorporated into the EM algorithm as part of this coursework. The brain tissue segmentation task will be applied, as in the prior assignment, and an atlas with the three probability maps for each tissue (WM, GM, and CSF) will be used.

For this task we used the atlas we've built on our own (part A of the assignment) for the atlas integration, and compared the results with those we got using the well-known MNI atlas template (given in the moodle). The implemented algorithm should work under the assumption of using model mixtures of Gaussians. The evaluation was done based on DSC scores for each of the tissues, for each of the patients, and for each of the different combinations' segmentation algorithms.

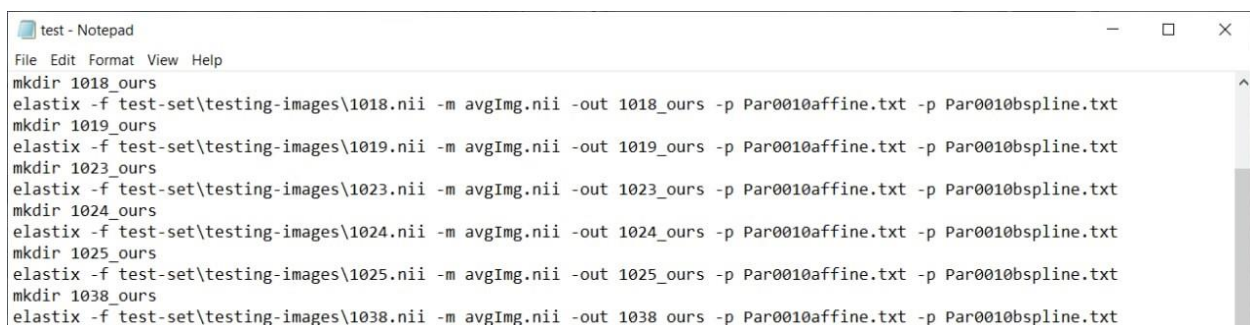
## ALGORITHM ANALYSIS

### *Registration*

The set of the brain volumes contains labels of three classes (WM, GM and CSF). The first step of implementation is to register the images for both of the atlases (our atlas and MNI given atlas). For that we used elastix, an open source toolbox for rigid and nonrigid registration of images(<http://elastix.isi.uu.nl/>).The parameter file we used is *Par0010* (<https://elastix.lumc.nl/modelzoo/par0010/>). The commands used for the registration and restoration are *elastix* and *transformix*. In order to visualize the registration, itk-Snap was used to illustrate that the registration worked as expected for rigid and nonrigid cases.

### 1) *Our Atlas*

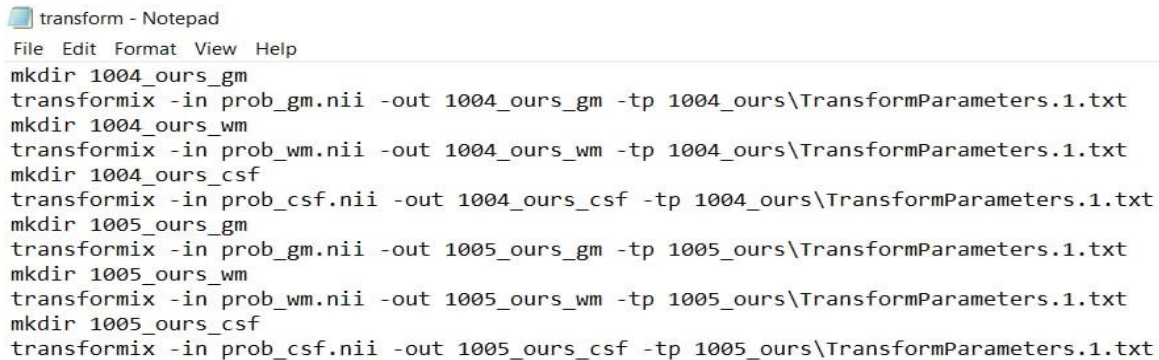
In this case the fixed image will be all the images found in the test-set folder. The moving image will be the avgImg.nii atlas we built in the last lab. Below we can see how we did the registration of the images using the average intensity atlas.



```
test - Notepad
File Edit Format View Help
mkdir 1018_ours
elastix -f test-set\testing-images\1018.nii -m avgImg.nii -out 1018_ours -p Par0010affine.txt -p Par0010bspline.txt
mkdir 1019_ours
elastix -f test-set\testing-images\1019.nii -m avgImg.nii -out 1019_ours -p Par0010affine.txt -p Par0010bspline.txt
mkdir 1023_ours
elastix -f test-set\testing-images\1023.nii -m avgImg.nii -out 1023_ours -p Par0010affine.txt -p Par0010bspline.txt
mkdir 1024_ours
elastix -f test-set\testing-images\1024.nii -m avgImg.nii -out 1024_ours -p Par0010affine.txt -p Par0010bspline.txt
mkdir 1025_ours
elastix -f test-set\testing-images\1025.nii -m avgImg.nii -out 1025_ours -p Par0010affine.txt -p Par0010bspline.txt
mkdir 1038_ours
elastix -f test-set\testing-images\1038.nii -m avgImg.nii -out 1038_ours -p Par0010affine.txt -p Par0010bspline.txt
```

**Figure 1:** Registered images using avgImg.nii as moving image

The output .txt file is the input .txt file for the transformix command. In this case the input image will be each of the tissue probability maps generated in the previous lab.



```
transform - Notepad
File Edit Format View Help
mkdir 1004_ours_gm
transformix -in prob_gm.nii -out 1004_ours_gm -tp 1004_ours\TransformParameters.1.txt
mkdir 1004_ours_wm
transformix -in prob_wm.nii -out 1004_ours_wm -tp 1004_ours\TransformParameters.1.txt
mkdir 1004_ours_csf
transformix -in prob_csf.nii -out 1004_ours_csf -tp 1004_ours\TransformParameters.1.txt
mkdir 1005_ours_gm
transformix -in prob_gm.nii -out 1005_ours_gm -tp 1005_ours\TransformParameters.1.txt
mkdir 1005_ours_wm
transformix -in prob_wm.nii -out 1005_ours_wm -tp 1005_ours\TransformParameters.1.txt
mkdir 1005_ours_csf
transformix -in prob_csf.nii -out 1005_ours_csf -tp 1005_ours\TransformParameters.1.txt
```

**Figure 2:** Registered tissues using TransformParameters.1.txt as input file

## 2) MNI Atlas

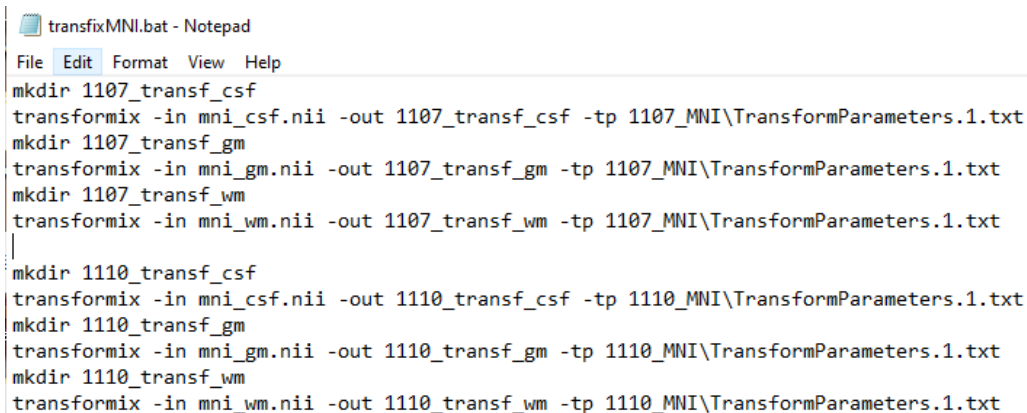
Another registration we did was using the atlas provided in moodle. The first part of the registration was done using elastix as a command. The fixed images in this case are all the images in the test-set folder as before. The moving image is template.nii provided within the coursework. Below is visualized how the registration of the images using the MNI template atlas was done.



```
testMNI.bat - Notepad
File Edit Format View Help
mkdir 1107_MNI
elastix -f test-set\testing-images\1107.nii -m template.nii -out 1107_MNI -p Par0010affine.txt -p Par0010bspline.txt
mkdir 1110_MNI
elastix -f test-set\testing-images\1110.nii -m template.nii -out 1110_MNI -p Par0010affine.txt -p Par0010bspline.txt
mkdir 1113_MNI
elastix -f test-set\testing-images\1113.nii -m template.nii -out 1113_MNI -p Par0010affine.txt -p Par0010bspline.txt
mkdir 1116_MNI
elastix -f test-set\testing-images\1116.nii -m template.nii -out 1116_MNI -p Par0010affine.txt -p Par0010bspline.txt
```

**Figure 3:** Registered images using template.nii as moving image

Before moving to the registration of each tissue, splitting the atlas.nii into each of the tissues since the image is a 4D volume and we need to work in separate volumes. The splitting was done in matlab. The resulting files are mni\_csf.nii, mni\_wm.nii and mni\_gm.nii. The next step is to use the transformix command and take as input images each of these resulting files.



```
transfixMNI.bat - Notepad
File Edit Format View Help
mkdir 1107_transf_csf
transformix -in mni_csf.nii -out 1107_transf_csf -tp 1107_MNI\TransformParameters.1.txt
mkdir 1107_transf_gm
transformix -in mni_gm.nii -out 1107_transf_gm -tp 1107_MNI\TransformParameters.1.txt
mkdir 1107_transf_wm
transformix -in mni_wm.nii -out 1107_transf_wm -tp 1107_MNI\TransformParameters.1.txt
mkdir 1110_transf_csf
transformix -in mni_csf.nii -out 1110_transf_csf -tp 1110_MNI\TransformParameters.1.txt
mkdir 1110_transf_gm
transformix -in mni_gm.nii -out 1110_transf_gm -tp 1110_MNI\TransformParameters.1.txt
mkdir 1110_transf_wm
transformix -in mni_wm.nii -out 1110_transf_wm -tp 1110_MNI\TransformParameters.1.txt
```

**Figure 4:** Registered tissues using TransformParameters.1.txt as input file for each patient

### *Segmentation*

After having the registered atlases, the following different segmentation methods were implemented:

- Segmentation using the tissue models computed from the previous lab sessions.
- Segmentation using label propagation from the registered atlas previously mentioned.
- Segmentation using both tissue models and label propagation, by multiplying the probabilities obtained from them.
- Segmentation using EM algorithm, with different initializations:
  - Tissue model initialization
  - Label propagation initialization
  - K-means initialization
- Segmentation with EM with the best initialization out of the previous ones, while feeding it our registered atlas information into the algorithm by multiplying it with the weights after each loop.
- Segmentation with EM with tissue model + label propagation initialization, while feeding it our registered atlas information into the algorithm by multiplying it with the weights after each loop.
- Segmentation with EM with tissue model + label propagation initialization, while feeding it the registered MNI atlas information into the algorithm by multiplying it with the weights after each loop.

### **DESIGN AND IMPLEMENTATION**

For implementing all the different segmentation methods, the atlas with the tissue models and the probability maps was integrated with the EM algorithm from the previous lab sessions.

To start, a data frame was created, with the sorted file names obtained from the different Google Drive folders containing all of our files. Figures 5 and 6 show the code for creating the data frame.

```

## Initialize arrays for file names
fouratlas_array = np.empty(0)
fourcsf_array = np.empty(0)
fourgm_array = np.empty(0)
fourwm_array = np.empty(0)

fmniatlas_array = np.empty(0)
fmnicssf_array = np.empty(0)
fmnigm_array = np.empty(0)
fmniwm_array = np.empty(0)

fimg_array = np.empty(0)
fmask_array = np.empty(0)
flabels_array = np.empty(0)

## Sort the atlas files
for i in os.listdir(path_our_atlas):
    four_atlas = i
    fouratlas_array = np.append(fouratlas_array, four_atlas)
    fouratlas_array = fouratlas_array.sort()

for i in os.listdir(path_fmni_atlas):
    fmni_atlas = i
    fmniatlas_array = np.append(fmniatlas_array, fmni_atlas)
    fmniatlas_array = fmniatlas_array.sort()

## Divide atlas files array into separate tissue arrays
idx = 0
while idx < 58:
    four_csف = fouratlas_array[idx]
    four_gm = fouratlas_array[idx+1]
    four_wm = fouratlas_array[idx+2]

    fmni_csف = fmniatlas_array[idx]
    fmni_gm = fmniatlas_array[idx+1]
    fmni_wm = fmniatlas_array[idx+2]

    fourcsf_array = np.append(fourcsf_array, four_csف)
    fourgm_array = np.append(fourgm_array, four_gm)
    fourwm_array = np.append(fourwm_array, four_wm)

    fmnicssf_array = np.append(fmnicssf_array, fmni_csف)
    fmnigm_array = np.append(fmnigm_array, fmni_gm)
    fmniwm_array = np.append(fmniwm_array, fmni_wm)

    idx += 3

## Sort the image, labels and mask files
for i, j, k in zip(os.listdir(path_test_images), os.listdir(path_test_masks), os.listdir(path_test_labels)):
    fimg = i
    fmask = j
    flabels = k

    fimg_array = np.append(fimg_array, fimg)
    fmask_array = np.append(fmask_array, fmask)
    flabels_array = np.append(flabels_array, flabels)

    fimg_array = np.sort(fimg_array)
    fmask_array = np.sort(fmask_array)
    flabels_array = np.sort(flabels_array)

## Create dataframe with filenames arrays
dict = {'fourcsf_array': fourcsf_array, 'fourgm_array': fourgm_array, \
        'fourwm_array': fourwm_array, 'fmnicssf_array': fmnicssf_array, \
        'fmnigm_array': fmnigm_array, 'fmniwm_array': fmniwm_array, \
        'fimg_array': fimg_array, 'fmask_array': fmask_array, 'flabels_array': flabels_array}
df_arrays = pd.DataFrame(dict)

```

Figure 5: Array initialization and Dataframe

Additionally, the tissue models computed from the previous lab had to be prepared, turning the NaNs into 0's and extending the end of the arrays to account for higher intensities that could probably appear in the test images. Figure 6 shows the tissue models preparation.

```

## Clean NaNs
csf_total_model = np.nan_to_num(csf_total.copy())
gm_total_model = np.nan_to_num(gm_total.copy())
wm_total_model = np.nan_to_num(wm_total.copy())

## Make intensities higher than 1500 into WM
csf_total_model[1500:] = 0
gm_total_model[1500:] = 0
wm_total_model[1500:] = 1

## Make array bigger in case of higher intensities and assign them to WM
end_array_csf = np.full(4000, 0)
end_array_gm = np.full(4000, 0)
end_array_wm = np.full(4000, 1)
csf_total_model = np.concatenate((csf_total_model, end_array_csf), axis=None)
gm_total_model = np.concatenate((gm_total_model, end_array_gm), axis=None)
wm_total_model = np.concatenate((wm_total_model, end_array_wm), axis=None)

## Stack all tissue models
tissue_models_total = np.vstack((csf_total_model, gm_total_model, wm_total_model))

## Get the argmax for the probabilities
intensity_labels = np.argmax(tissue_models_total, axis=0) + 1

```

Figure 6: Tissue model preparation

Moreover, the EM algorithm implemented during the previous labs was turned into several function, in order to optionally implement it after the initial segmentation with the tissue models

and label propagation. The 3 different EM functions are called: *EM\_algorithm()*, *EM\_algorithm\_add\_atlas*, and *EM\_algorithm\_kmeans()*. The first is the basic function that accepts the labeled images coming from the Tissue Models or Label Propagation segmentation. The second accepts additional inputs from the probability maps from either our atlas or the MNI atlas. The third one doesn't take any labeled images coming from the previously mentioned segmentations and initializes the algorithm with k-means instead. Ideally, this would have been implemented in one single function, but due to time constraints it was implemented this way. Figure 7 shows the basic *EM\_algorithm* function.

```
def EM_algorithm(test_img, labeled_img, test_mask):
    mask_flat = test_mask.reshape(-1)
    skull_stripped = test_img * test_mask

    skull_stripped_flat = skull_stripped.reshape(-1)
    labeled_image_flat = labeled_img.reshape(-1)

    # Original indexes for reconstruction and flattened masked volume
    idx = np.where(mask_flat != 0)
    idx = idx[0] # idx has two elements (array[...], ), hence idx[0]
    skull_stripped_mask = skull_stripped_flat[mask_flat != 0] #####
    labeled_image_mask = labeled_image_flat[mask_flat != 0]

    # INITIAL PARAMETERS
    #alphas
    csf_count = np.count_nonzero(labeled_img == 1)
    gm_count = np.count_nonzero(labeled_img == 2)
    wm_count = np.count_nonzero(labeled_img == 3)
    bg_count = np.count_nonzero(labeled_img == 0)
    total_count = csf_count + gm_count + wm_count + bg_count

    a1 = csf_count/total_count
    a2 = gm_count/total_count
    a3 = wm_count/total_count

    #means
    mu_CSF = np.mean(skull_stripped_mask[labeled_image_mask == 1])
    mu_GM = np.mean(skull_stripped_mask[labeled_image_mask == 2])
    mu_WM = np.mean(skull_stripped_mask[labeled_image_mask == 3])
    #covariances
    cov_CSF = np.cov(skull_stripped_mask[labeled_image_mask == 1])
    cov_GM = np.cov(skull_stripped_mask[labeled_image_mask == 2])
    cov_WM = np.cov(skull_stripped_mask[labeled_image_mask == 3])

    # For Stopping criteria
    log_diff = 1000
    log_past = 0

    # EM LOOP
    while log_diff > 0.02:
        # Gaussian mixture model
        p1 = stats.multivariate_normal.pdf(skull_stripped_mask, mean=mu_CSF, cov=cov_CSF)
        p2 = stats.multivariate_normal.pdf(skull_stripped_mask, mean=mu_GM, cov=cov_GM)
        p3 = stats.multivariate_normal.pdf(skull_stripped_mask, mean=mu_WM, cov=cov_WM)

        # Membership weights
        w1 = (p1*a1)/(p1*a1+p2*a2+p3*a3)
        w2 = (p2*a2)/(p1*a1+p2*a2+p3*a3)
        w3 = (p3*a3)/(p1*a1+p2*a2+p3*a3)

        # New alphas
        N1 = np.sum(w1)
        N2 = np.sum(w2)
        N3 = np.sum(w3)
        a1 = N1/len(w1)
        a2 = N2/len(w2)
        a3 = N3/len(w3)

        # New means
        mu_CSF = (1/N1) * np.dot(w1, skull_stripped_mask)
        mu_GM = (1/N2) * np.dot(w2, skull_stripped_mask)
        mu_WM = (1/N3) * np.dot(w3, skull_stripped_mask)

        # New covariances
        subs1 = np.subtract(skull_stripped_mask, mu_CSF)
        subs1_transpose = subs1.transpose()
        subs2 = np.subtract(skull_stripped_mask, mu_GM)
        subs2_transpose = subs2.transpose()
        subs3 = np.subtract(skull_stripped_mask, mu_WM)
        subs3_transpose = subs3.transpose()

        cov_CSF = (1/N1) * np.dot(w1, subs1*subs1_transpose)
        cov_GM = (1/N2) * np.dot(w2, subs2*subs2_transpose)
        cov_WM = (1/N3) * np.dot(w3, subs3*subs3_transpose)

    # Log likelihood
    log_current = np.sum(np.log((a1*p1)+(a2*p2)+(a3*p3)))
    log_diff = abs(log_current - log_past)
    log_past = log_current
    # print('Log likelihood diff: ' + str(log_diff))

    # Final weights and labels
    weights = np.vstack((w1, w2, w3))
    final_labels = np.argmax(weights, axis=0)
    final_labels = final_labels + 1

    # Labels reshaped
    f_reconstruction = np.zeros_like(skull_stripped_flat)
    f_reconstruction[idx] = final_labels
    labeled_image = np.array(f_reconstruction).reshape(skull_stripped.shape)
    return labeled_image
```

Figure 7: EM algorithm function

Having done this, the next part was to implement the different segmentation algorithms. For the tissue models, we first read the needed files, and we segmented the test image with the labels

according to the pixel intensity obtained from the tissue models from the previous lab. This is done by using these intensity labels as a LUT for the test image. After this, there is an optional EM step with the function described above. Finally, we compute the DICE scores with the *dice()* function implemented in the previous lab. Figure 8 shows the Tissue Model Segmentation code.

```
for idx, row in df_arrays.iterrows():

    test_image = nib.load(os.path.join(path_test_images, row["fimg_array"])).get_fdata()
    test_mask = nib.load(os.path.join(path_test_masks, row["fmask_array"])).get_fdata()
    test_labels = nib.load(os.path.join(path_test_labels, row["flabels_array"])).get_fdata()
    our_atlas_csf = nib.load(os.path.join(path_mni_atlas, row["fmnicsf_array"])).get_fdata()
    our_atlas_gm = nib.load(os.path.join(path_mni_atlas, row["fmnigm_array"])).get_fdata()
    our_atlas_wm = nib.load(os.path.join(path_mni_atlas, row["fmniwm_array"])).get_fdata()

    ## SEGMENTING
    test_image_int = test_image.astype(int).copy()
    labeled_image = intensity_labels[test_image_int]
    labeled_image = labeled_image * test_mask #

    ## Optional EM
    if EM_flag == 1:
        #labeled_image = EM_algorithm(test_image, labeled_image, test_mask)
        labeled_image = EM_algorithm_add_atlas(test_image, labeled_image, test_mask, our_atlas_csf, our_atlas_gm, our_atlas_wm)

    ## DICE SCORES
    dice_CSF = dice(labeled_image, test_labels)[0]
    dice_GM = dice(labeled_image, test_labels)[1]
    dice_WM = dice(labeled_image, test_labels)[2]

    TM_csf_scores.append(dice_CSF)
    TM_gm_scores.append(dice_GM)
    TM_wm_scores.append(dice_WM)

    print(f'Image: {row["fimg_array"][:-4]}')
    print(f'CSF DICE: {dice_CSF}')
    print(f'GM DICE: {dice_GM}')
    print(f'WM DICE: {dice_WM} \n')
```

**Figure 8:** Tissue model Segmentation

Implementing the segmentation algorithm with label propagation was done similarly. The files were read, and the actual segmentation was done by stacking the registered probability maps from our atlas, and getting their argmax, thus obtaining the label for the tissue with the highest probability according to the pixel location. The optional EM step was added, and the DICE scores were finally computed. Figure 9 shows this part of the code.

For the tissue models + label propagation segmentation, we simply multiplied the probability maps for each tissue with the probability for each pixel to belong to each tissue with the tissue models. Afterwards, we stacked the result of these multiplications and obtained the argmax to get the labels for each pixel. The optional EM step is added then, and the DICE scores are finally computed. Figure 10 shows this part of the code.



```

for idx, row in df_arrays.iterrows():

    ## Reading files
    our_atlas_csf = nib.load(os.path.join(path_our_atlas, row["fourcsf_array"])).get_fdata()
    our_atlas_gm = nib.load(os.path.join(path_our_atlas, row["fourgm_array"])).get_fdata()
    our_atlas_wm = nib.load(os.path.join(path_our_atlas, row["fourwm_array"])).get_fdata()
    test_image = nib.load(os.path.join(path_test_images, row["fimg_array"])).get_fdata()
    mask = nib.load(os.path.join(path_test_masks, row["fmask_array"])).get_fdata()
    labels = nib.load(os.path.join(path_test_labels, row["flabels_array"])).get_fdata()

    # Stacking and getting the argmax
    our_atlas_total = np.stack((our_atlas_csf, our_atlas_gm, our_atlas_wm), axis=-1)
    position_labels = np.argmax(our_atlas_total, axis=-1) + 1
    position_labels = position_labels * mask #

    ## Optional EM
    if EM_flag == 1:
        position_labels = EM_algorithm(test_image, position_labels, mask)

    ## DICE SCORES
    dice_CSF = dice(position_labels, labels)[0]
    dice_GM = dice(position_labels, labels)[1]
    dice_WM = dice(position_labels, labels)[2]

    LP_csf_scores.append(dice_CSF)
    LP_gm_scores.append(dice_GM)
    LP_wm_scores.append(dice_WM)

    print(f'Image: {row["fimg_array"][:-4]}')
    print(f'CSF DICE: {dice_CSF}')
    print(f'GM DICE: {dice_GM}')
    print(f'WM DICE: {dice_WM} \n')

```

Figure 9: Label Propagation Segmentation

```

for idx, row in df_arrays.iterrows():

    ## Reading files
    our_atlas_csf = nib.load(os.path.join(path_our_atlas, row["fourcsf_array"])).get_fdata()
    our_atlas_gm = nib.load(os.path.join(path_our_atlas, row["fourgm_array"])).get_fdata()
    our_atlas_wm = nib.load(os.path.join(path_our_atlas, row["fourwm_array"])).get_fdata()
    test_image = nib.load(os.path.join(path_test_images, row["fimg_array"])).get_fdata()
    mask = nib.load(os.path.join(path_test_masks, row["fmask_array"])).get_fdata()
    labels = nib.load(os.path.join(path_test_labels, row["flabels_array"])).get_fdata()
    test_image_int = test_image.astype(int).copy()

    ## SEGMENTING
    prob_csf_model = csf_total_model[test_image_int]
    prob_gm_model = gm_total_model[test_image_int]
    prob_wm_model = wm_total_model[test_image_int]
    both_csf = np.multiply(prob_csf_model, our_atlas_csf)
    both_gm = np.multiply(prob_gm_model, our_atlas_gm)
    both_wm = np.multiply(prob_wm_model, our_atlas_wm)
    both_total = np.stack((both_csf, both_gm, both_wm), axis=-1)
    both_labels = np.argmax(both_total, axis=-1) + 1
    both_labels = both_labels * mask #

    ## Optional EM
    if EM_flag == 1:
        both_labels = EM_algorithm_add_atlas(test_image, both_labels, mask, our_atlas_csf, our_atlas_gm, our_atlas_wm)

    ## DICE SCORES
    dice_CSF = dice(both_labels, labels)[0]
    dice_GM = dice(both_labels, labels)[1]
    dice_WM = dice(both_labels, labels)[2]

    BM_csf_scores.append(dice_CSF)
    BM_gm_scores.append(dice_GM)
    BM_wm_scores.append(dice_WM)

    print(f'Image: {row["fimg_array"][:-4]}')
    print(f'CSF DICE: {dice_CSF}')
    print(f'GM DICE: {dice_GM}')
    print(f'WM DICE: {dice_WM} \n')

```

Figure 10: Tissue model Segmentation



As for segmenting with the EM algorithm initialized with K-means, we read the files as in the previous steps, and then we implemented the EM algorithm with the k-means initialization function implemented in the previous lab. The labels are obtained and used for computing the DICE scores. Figure 11 shows the code for segmentation with k-means.

```
KM_csf_scores = []
KM_gm_scores = []
KM_wm_scores = []

for idx, row in df_arrays.iterrows():

    test_image = nib.load(os.path.join(path_test_images, row["fimg_array"])).get_fdata()
    test_mask = nib.load(os.path.join(path_test_masks, row["fmask_array"])).get_fdata()
    test_labels = nib.load(os.path.join(path_test_labels, row["flabels_array"])).get_fdata()

    kmeans_labels = EM_algorithm_kmeans(test_image, test_mask, flag='kmeans')

    ## DICE SCORES
    dice_CSF = dice(kmeans_labels, test_labels)[0]
    dice_GM = dice(kmeans_labels, test_labels)[1]
    dice_WM = dice(kmeans_labels, test_labels)[2]

    KM_csf_scores.append(dice_CSF)
    KM_gm_scores.append(dice_GM)
    KM_wm_scores.append(dice_WM)

    print(f'Image: {row["fimg_array"][:-4]}')
    print(f'CSF DICE: {dice_CSF}')
    print(f'GM DICE: {dice_GM}')
    print(f'WM DICE: {dice_WM} \n')
```

Figure 11: K-Means Segmentation

Finally, to save the csv files and create the box plots for visualizing the DICE scores for each of the algorithms, we initialized a list for each of the tissues before each method and appended the computed DICE scores after every loop. This part is highlighted in red in Figure 11.

The final part of the code consists of saving the csv files and visualizing the plots, which is done by creating a list of lists with the DICE scores for all the tissues computed with a given method, and then saving that with the `np.savetxt()` function. Finally, we used seaborn to visualize the box plots. Figure 12 shows this implementation.

```

Names = ['CSF', 'GM', 'WM']

TM = []
TM.append(TM_csf_scores)
TM.append(TM_gm_scores)
TM.append(TM_wm_scores)

LP = []
LP.append(LP_csf_scores)
LP.append(LP_gm_scores)
LP.append(LP_wm_scores)

BM = []
BM.append(BM_csf_scores)
BM.append(BM_gm_scores)
BM.append(BM_wm_scores)

KM = []
KM.append(KM_csf_scores)
KM.append(KM_gm_scores)
KM.append(KM_wm_scores)

np.savetxt("TM.csv", np.transpose(TM), delimiter=",", fmt = '% s')
np.savetxt("LP.csv", np.transpose(LP), delimiter=",", fmt = '% s')
np.savetxt("BM.csv", np.transpose(BM), delimiter=",", fmt = '% s')
np.savetxt("KM.csv", np.transpose(KM), delimiter=",", fmt = '% s')

import seaborn as sns

plt.boxplot(TM, labels = Names)
plt.title('Tisse Models (EM + MNIAAtlas) Dice Score')
plt.ylabel('Dice Score')
sns.set_style("whitegrid")
plt.ylim(0.1,1)
plt.grid('-')
plt.show()

```

**Figure 12:** Visualization and CSV saving files algorithms

## EXPERIMENTS AND RESULTS

Table 1 shows the DICE scores computed for each of the algorithms, per tissue, per case. The highest mean score for each tissue amongst all of the models is highlighted.

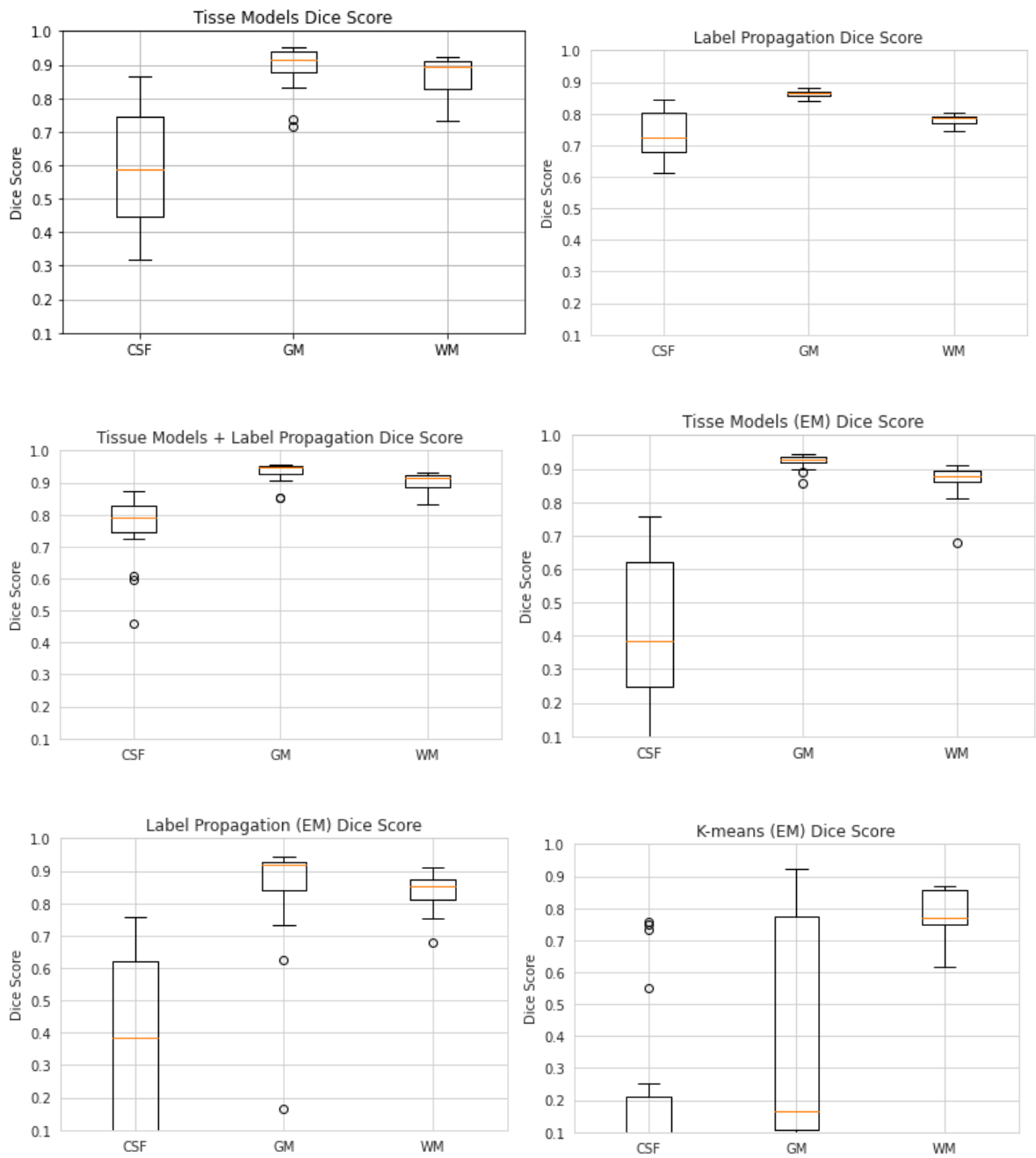
TABLE 1: Dice score per algorithm

	TISSUE MODELS			LABEL PROPAGATION			TM + LP			TM (EM)			LP (EM)		
	CSF	GM	WM	CSF	GM	WM	CSF	GM	WM	CSF	GM	WM	CSF	GM	WM
1003	0.3172	0.8306	0.8109	0.6104	0.8672	0.7867	0.5944	0.9047	0.8777	0.1112	0.9356	0.8923	0.0001	0.9225	0.7530
1004	0.6160	0.9465	0.9210	0.8035	0.8628	0.7892	0.8370	0.9560	0.9322	0.4064	0.9339	0.8966	0.4064	0.9339	0.8966
1005	0.8647	0.9459	0.9192	0.7006	0.8395	0.7481	0.8727	0.9469	0.9130	0.7487	0.9213	0.8675	0.7487	0.9213	0.8675
1018	0.4640	0.9131	0.8892	0.8223	0.8647	0.7841	0.7731	0.9493	0.9273	0.2597	0.9336	0.8948	0.0196	0.7855	0.8470
1019	0.4181	0.9505	0.9181	0.6493	0.8588	0.7689	0.7798	0.9565	0.9241	0.0978	0.9413	0.8949	0.0362	0.7310	0.8357
1023	0.4068	0.9043	0.8790	0.6155	0.8611	0.7768	0.7235	0.9441	0.9210	0.2101	0.9334	0.8908	0.0265	0.7391	0.8413
1024	0.5681	0.9318	0.9009	0.7775	0.8719	0.7882	0.8103	0.9555	0.9304	0.3532	0.9415	0.9016	0.3532	0.9415	0.9016
1025	0.8519	0.9449	0.9097	0.6820	0.8456	0.7454	0.8586	0.9474	0.9059	0.7315	0.9222	0.8620	0.7315	0.9222	0.8620
1038	0.4585	0.9322	0.9108	0.8436	0.8642	0.7856	0.7625	0.9537	0.9315	0.2717	0.9338	0.8989	0.2717	0.9338	0.8989
1039	0.3506	0.9422	0.9164	0.6724	0.8552	0.7714	0.7465	0.9550	0.9283	0.0888	0.9355	0.8883	0.0328	0.6265	0.8028
1101	0.6875	0.9305	0.9050	0.8005	0.8664	0.7790	0.8008	0.9500	0.9235	0.5490	0.9224	0.8671	0.5490	0.9224	0.8671
1104	0.6067	0.9005	0.8626	0.8100	0.8676	0.7880	0.7635	0.9356	0.9024	0.4268	0.9323	0.8808	0.4268	0.9323	0.8808
1107	0.3280	0.7147	0.7319	0.8060	0.8815	0.7973	0.4590	0.8545	0.8300	0.0000	0.9187	0.8610	0.0460	0.1647	0.7844
1110	0.5705	0.9157	0.8828	0.7926	0.8741	0.7944	0.7388	0.9473	0.9178	0.3654	0.9430	0.9112	0.3654	0.9430	0.9112
1113	0.4889	0.7367	0.7596	0.7994	0.8678	0.8019	0.6076	0.8530	0.8386	0.3089	0.9029	0.8354	0.0000	0.8855	0.7949
1116	0.7286	0.9360	0.9152	0.7097	0.8502	0.7606	0.8409	0.9468	0.9151	0.6147	0.9178	0.8659	0.6147	0.9178	0.8659
1119	0.8023	0.8502	0.8316	0.6817	0.8425	0.7651	0.8048	0.9091	0.8843	0.6841	0.8900	0.8130	0.6841	0.8900	0.8130
1122	0.7947	0.8326	0.8140	0.7402	0.8614	0.7897	0.8224	0.9073	0.8802	0.6422	0.9163	0.8722	0.6422	0.9163	0.8722
1125	0.8643	0.9167	0.8951	0.6635	0.8533	0.7719	0.8419	0.9358	0.9099	0.7587	0.8996	0.8366	0.7587	0.8996	0.8366
1128	0.6819	0.8876	0.7922	0.7028	0.8673	0.7868	0.8125	0.9319	0.8811	0.5768	0.8558	0.6786	0.5768	0.8558	0.6786
Mean	0.5935	0.8932	0.8682	0.7342	0.8612	0.7790	0.7625	0.9320	0.9037	0.4103	0.9215	0.8655	0.3645	0.8392	0.8406
Std	0.1805	0.0662	0.0564	0.0718	0.0105	0.0150	0.0995	0.0305	0.0290	0.2359	0.0207	0.0494	0.2821	0.1762	0.0556

TABLE 1: Dice score per algorithm

	K-MEANS EM			TM (EM + OurAtlas)			TM + LP (EM + OurAtlas)			(EM + MNIAtlas)		
	CSF	GM	WM	CSF	GM	WM	CSF	GM	WM	CSF	GM	WM
1003	0.0263	0.1662	0.7530	0.4085	0.9543	0.9247	0.4085	0.9543	0.9247	0.0552	0.8866	0.7976
1004	0.0352	0.0859	0.7316	0.6995	0.9599	0.9401	0.6995	0.9599	0.9401	0.1568	0.8854	0.8187
1005	0.7487	0.9213	0.8675	0.7868	0.9485	0.9226	0.7868	0.9485	0.9226	0.5408	0.8836	0.8036
1018	0.0353	0.0802	0.7688	0.6609	0.9562	0.9310	0.6609	0.9562	0.9310	0.1722	0.8895	0.8141
1019	0.0248	0.1826	0.7732	0.4349	0.9625	0.9356	0.4349	0.9625	0.9356	0.0484	0.9011	0.8240
1023	0.0243	0.1170	0.7609	0.4767	0.9541	0.9274	0.4767	0.9541	0.9274	0.0631	0.8896	0.8136
1024	0.0310	0.1198	0.7522	0.6489	0.9641	0.9421	0.6489	0.9641	0.9421	0.1327	0.8997	0.8288
1025	0.7315	0.9222	0.8620	0.7661	0.9482	0.9170	0.7661	0.9482	0.9170	0.5260	0.8853	0.7991
1038	0.0245	0.7728	0.8523	0.6468	0.9557	0.9319	0.6468	0.9557	0.9319	0.1836	0.8896	0.8191
1039	0.0268	0.1477	0.7192	0.4569	0.9580	0.9296	0.4569	0.9580	0.9296	0.0519	0.8893	0.8048
1101	0.5490	0.9224	0.8671	0.6901	0.9460	0.9116	0.6901	0.9460	0.9116	0.3111	0.8894	0.8111
1104	0.0422	0.1508	0.7914	0.6745	0.9546	0.9261	0.6745	0.9546	0.9261	0.2463	0.9080	0.8444
1107	0.0460	0.1647	0.7844	0.4210	0.9370	0.8900	0.4210	0.9370	0.8900	0.1287	0.8754	0.7684
1110	0.0546	0.0711	0.6146	0.6172	0.9635	0.9422	0.6172	0.9635	0.9422	0.2476	0.9049	0.8384
1113	0.0551	0.0574	0.7110	0.5492	0.9442	0.9143	0.5492	0.9442	0.9143	0.2230	0.8702	0.7799
1116	0.2520	0.7783	0.8660	0.6988	0.9426	0.9100	0.6988	0.9426	0.9100	0.4073	0.8855	0.8117
1119	0.1215	0.0125	0.7371	0.7417	0.9303	0.8979	0.7417	0.9303	0.8979	0.4984	0.8575	0.7657
1122	0.1442	0.3810	0.8707	0.7300	0.9419	0.9160	0.7300	0.9419	0.9160	0.0727	0.0117	0.0891
1125	0.7587	0.8996	0.8366	0.7664	0.9302	0.9021	0.7664	0.9302	0.9021	0.5747	0.8748	0.8045
1128	0.1988	0.6308	0.7537	0.6653	0.9367	0.9035	0.6653	0.9367	0.9035	0.3981	0.8866	0.8336
Mean	0.1965	0.3792	0.7837	0.6270	0.9494	0.9208	0.6270	0.9494	0.9208	0.2519	0.8432	0.7735
Std	0.2603	0.3473	0.0666	0.1210	0.0103	0.0147	0.1210	0.0103	0.0147	0.1748	0.1911	0.1583

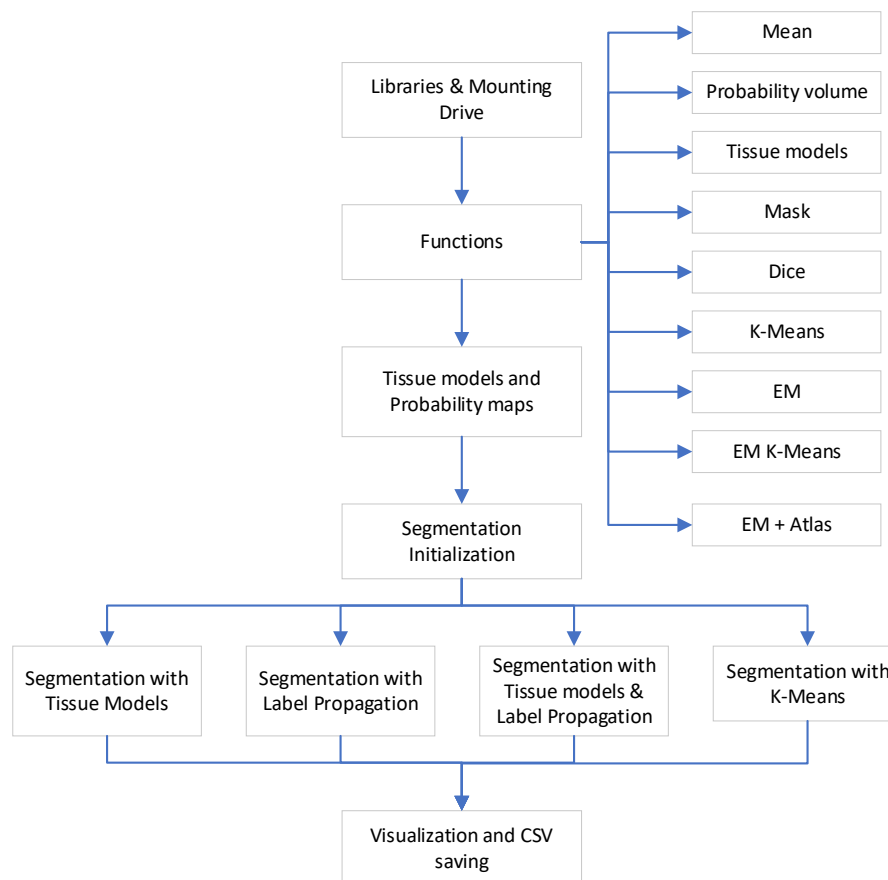
The box plots for these DICE scores are shown in the following Figures:



## COURSEWORK MANAGEMENT

The lab was divided into the following tasks:

- Research and reading
- Code implementation
  - Initializing the models
  - Creating the dataframe
  - Tissue Models Segmentation
  - Label Propagation Segmentation
  - Tissue Models + Label Propagations Segmentation
  - EM function
  - Combining EM with the different initializations
  - Plotting and saving CSVs



**Figure 13:** Code implementation Flowchart

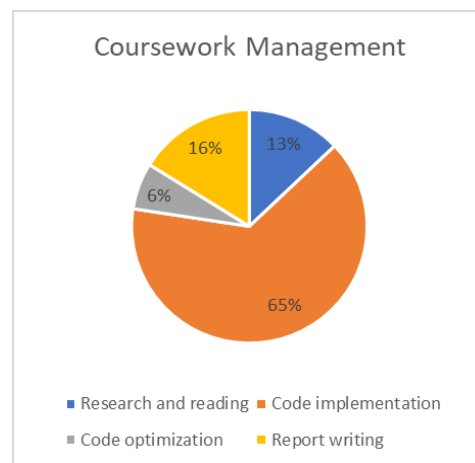
- Report writing

All of the team members contributed equally to the time spent on this work. We collaborated continuously both outside the campus and in the lab. Before implementing the code, each of us had to do some research to comprehend the problem and think of a solution. The tasks were

divided equally among the team members, but everyone helped in other tasks when necessary. We all collaborated in Google Colab for the code, and in Google Docs for the report.

The estimated total time for project completion was around a week. We expected to finish rather quickly, as the tasks seemed quite simple and we already had the composing parts of the code. However, there were some setbacks with unexpected bugs in the code that took more than what was planned, causing us to spend the whole week on this. The approximate time for each of the tasks was:

- Research and reading: 4 hours
- Code implementation: 20 hours
- Code optimization/debugging 2 hours
- Report writing: 5 hours



**Figure 14:** Coursework management distribution chart.

## CONCLUSIONS

A probabilistic atlas is a useful technique for registering new images and segmenting them. With the tools provided to us during the lab, registering the images proved to be quite simple, with the registered images performing as expected and moving into the same space as the fixed image. The probabilistic label volumes are a simple method for further segmenting additional volumes because, after registering them in the same space as the atlas, they are given labels based on the highest probability at each voxel. For this lab both atlases were taken into consideration.

Since CSF, GM, and WM have diverse intensity distributions, normalizing the histogram into probabilities can also aid in segmentation because it allows us to identify new volumes based on the intensity of a particular voxel. The comparison is done based on both registered images: using first our atlas as moving image and secondly using the MNI atlas provided in the course. In this way, a comparison between both methods is implied.

After running the experiments, we noticed that CSF DICE scores were always lower than the rest. This could be due to there existing more anatomical variability in patients in this particular tissue.

Tissue Models + Label Propagation had overall very high average scores for all three tissues. Even though both TM (EM + OurAtlas) and TM + LP (EM + OurAtlas) had higher GM and WM DICE SCORES, CSF's average score dropped, while GM and WM were not much higher.

The EM algorithm did not generally perform very well, mainly due to its inability to properly segment CSF. We don't know if this is due to a bug that we could not find, but generally, algorithms without EM worked better. K-means did particularly bad when compared to the rest, which could be caused by it having no prior information of the tissues, unlike with other initializations. The alphas are initialized as 1/3 and it has no prior location and intensity information.

Another interesting observation is that the DICE scores for two of the algorithms, TM (EM + OurAtlas) and TM + LP (EM + OurAtlas) were exactly the same. We assume that this could either be caused by an error in the implementation or by the algorithm reaching the same local minima in both situations.

## GOOGLE DRIVE FOLDERS

- Transformed images with our atlas

<https://drive.google.com/drive/u/1/folders/1pk6eyJxPonej1N1SiZ6tdFkKRQb8Cw-x>

- Images from (our) Probabilistic Atlas

[https://drive.google.com/drive/folders/13ebin1lrCysjdjSv32Jb9NCGy\\_BezMA7?usp=sharing](https://drive.google.com/drive/folders/13ebin1lrCysjdjSv32Jb9NCGy_BezMA7?usp=sharing)

- Transformed images with MNI atlas

<https://drive.google.com/drive/u/1/folders/1ePVVNIq6IV2Xz5ITfPVyZqRqpA3M7S0I>

## REFERENCES

- Chakravarty, M. & Bertrand, Gilles & Descoteaux, Maxime & Sadikot, Abbas & Collins, Louis. (2003). The Creation of a Brain Atlas for Image Guided Neurosurgery Using Serial Histological Data. 343-350. 10.1007/978-3-540-39899-8\_43.
- Cuadra, M.B., Pollo, C., Bardera, A., Cuisenaire, O., Villemure, J.G., & Thiran, J. (2004). Atlas-based segmentation of pathological MR brain images using a model of lesion growth. *IEEE Transactions on Medical Imaging*, 23, 1301-1314.
- Kalinić, H. (2009). Atlas-based image segmentation: A Survey.