HL2027 - Project 2
Segmentation and analysis of pelvic bone in CT images

G03: Rebecca Bonato and Lasse Stahnke

January 9, 2023

1 Introduction

The aim of the project consists in developing useful strategies for hip surgery planning starting from 3D CT images. In particular, the project is divided in two main parts. The first purpose consists in implementing an Atlas-based segmentation tool to segment pelvic bone and femur and to compare its performance with a standard segmentation method. The second part of the project aims to localize the Obturator foramen on sagital 2D slices since it plays a relevant role in surgery planning. To reach the goal, a classifier has to be trained in order to detect the slice with the highest probability of containing the obturator formamen in each 3D image.

2 Raw Data

The starting data are available at this link. In particular, we had access to two datasets:

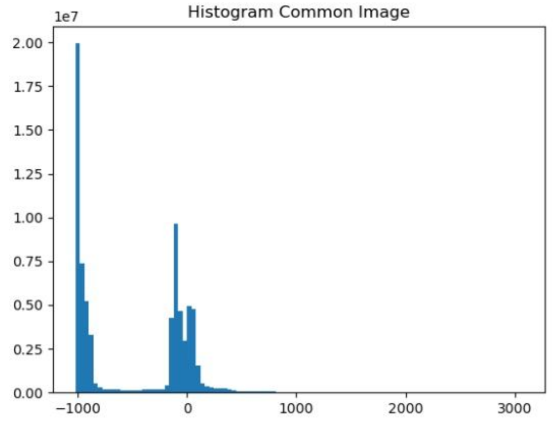
1. "GROUP_images": three 3D CT images belonging to the group one. These images will be used to create the atlas database in the segmentation part of the project and to train the classifier in the analysis part.

2. "COMMON_images_masks": three 3D CT images with their correspondent ground-truth masks for right and left femur, right and left hip bone and sacrum. On common image, we applied the atlas based segmentation. The ground truths were performed by an expert radiologist and we used them to test the accuracy of our segmentation (both with the standard method and the atlas based one). In the analysis part, the performance of the trained classifier, has been tested on these Common images.

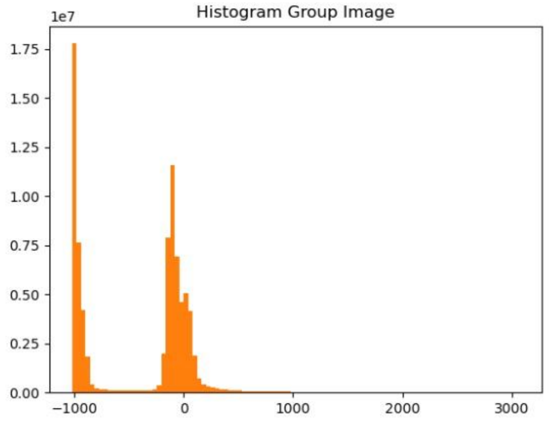
3 Methods

3.1 Manual segmentation

For the manual segmentation, a seeded region growing was used. Thereby, the seeds used were carefully placed between interfaces of tissues (i.e. between the femoral head and the pelvis as well as each bone and the background). The segmentations were done using 3D Slicer [1]. Slicer provides a preview of the segmentation. Using that preview, the seeds were refined successively. The manual segmentation was performed on all images in the "COMMON_images_masks" directory as well as all the images belonging to the first Group in the "GROUP_images" directory. After region growing a smoothing was applied to make the bones smooth and remove artifacts from the segmentation.



(a) Image from Common directory



(b) Image from Group directory

Figure 1: Histograms of one image per directory. It can be seen that the distributions of the grayscale values are similar to each other. This is intuitive as Hounsfield-units are usually used in CT imaging. Furthermore, this implies that a single threshold can be used for all images to mask bones in the images.

3.2 Atlas-Based Segmentation

For the atlas-based segmentation, a two-method registration approach was used. The images within the "GROUP_images" directories and their respective manual segmentations were used as an atlas. Thereby, the one that was supposed to be segmented was used as a reference image. All images within the atlas were registered onto that reference image. The whole pipeline consisted of the following steps:

1. Preprocessing
2. Linear registration
3. Non-Linear registration
4. Majority voting

For the whole pipeline, SimpleITK was used. The pipeline is analysed more in detail in the following subsections.

3.2.1 Preprocessing

In the case of non-linear-registration, simple thresholding was used on the reference and the moving images. The threshold was empirically chosen to a grayscale value of 150 to get rid of the background and retain the bones in the images.

3.2.2 Linear Registration

For the linear registration, a the SimpleITK method *CenteredTransformInitializer* was used. The method aligns the centers of the images using translations. As an optimizer, gradient descent has been used in combination with Mattes Mutual Information metric. To speed up the registration process only 10% of sampling points were used.

3.2.3 Non-linear Registration

For the non-linear method Bspline registration was used. Here, the *GradientDescentLineSearch* optimizer in combination with Mattes Mutual Information metric was chosen. In order to accelerate the registration, only 1 percent of image points were randomly sampled. The interpolation method used to apply the estimated transforms (both linear and non linear) is Nearest Neighbour: in that way, the mask's discrete values are not changed.

3.2.4 Majority Voting

Finally, to get the segmentation of the reference image, all estimated transforms were applied to their respective segmentation maps and were combined using majority voting to get the segmentation of the reference image.

3.2.5 Accuracy Assessment

To asses the accuracy of both the region growing based and the atlas based segmentation on common images, two metrics have been used:

1. Dice Coefficient: it is a measure of overlapping between the reference and the segmented mask.
2. Hausdorff Distance: since it is a distance measure, it's much more sensitive to noise than the Dice coefficient.

3.3 Slice Obturator foramen detection

This subsection of the project includes different steps that are enumerated below:

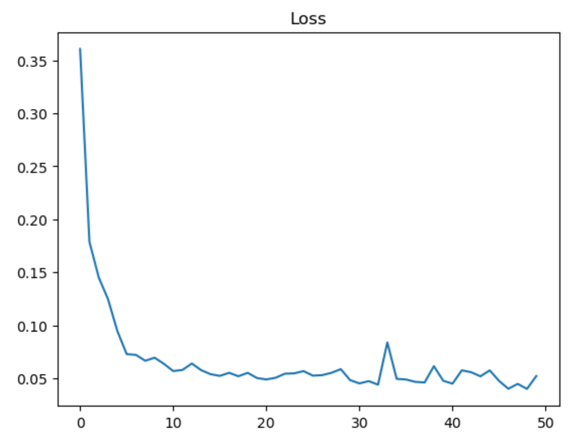
1. Manual classification of GROUP 2D sagital slices
2. Classifier selection
3. Preprocessing
4. Training
5. Obturator foramen detection on COMMON sagital slices

3.3.1 Manual classification

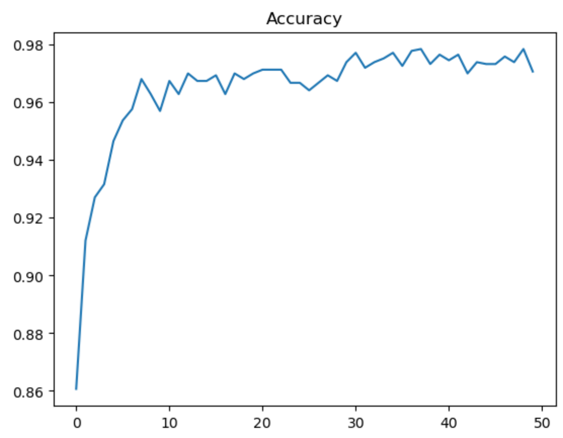
The manual classification of GROUP 2D sagital slices has been done exploiting itk-SNAP.

3.3.2 Classifier selection

For the classification of the Obturator foramen, a convolutional neural work was used. In particular, a 2D LeNet [2] architecture was chosen. Due to the small number of training images, the network architecture was modified. Dropout was used in the fully connected layer as well as batch normalization after each convolutional layer. The number of feature maps in the first convolutional layer was 8. Since we are dealing with a binary classification, it's also important to specify that the number of feature maps in the last dense layer of the network has been set as one and the activation function chosen is a sigmoid function. The code for the classification can be found on GitHub (`https://github.com/lassestahnke/femur_segmentation_atlas/blob/main/main_classification.ipynb`.



(a) Loss curve



(b) Accuracy curve

Figure 2: Learning curves of the classifier. Notice that the accuracy is already at 86% after the first epoch. There are around 86% of slices without the Obturator foramen. This means that 0.86 is the accuracy if the network only outputs 0 as predictions. It can also be seen that the network seems to have converged.

3.3.3 Preprocessing

The three images from the GROUP directory were split into sagital slices. Each slice was then resized to the shape of the slices of the first image to provide same-sized images to the network. This size is returned by the network training function to apply it to new images as well. Normalizing the images was considered by did not yield any substantial improvement in performance. Thus, it was excluded from further experiments. Furthermore, the images were thresholded to only show grayscale values above 100, to only consider bones and not noise in the background class.

3.3.4 Training

The network was trained on all GROUP images with a batch size of 32 (slices). 50 epochs were used. Since there is a high imbalance between slices that contain the Obturator foramen and ones that don't show it, binary cross-entropy was chosen as a loss function. Adam optimizer was used to optimize the network. To evaluate the performance, the accuracy metric was used. Due to the imbalance in traning data, there are around 86% slices that contain no Obturator foramen. Thus, if the network is not trained and only predicts zeros, the accuracy is already at 86%.

3.4 Testing

Finally, the trained model was used to predict the probability of containing the obturatur foramen on each COMMON image.

4 Results

4.1 Manual Segmentation

The segmentation results of the manual segmentation can be seen in Table 2. It can be seen that the DSC is around 0.95 for all images. It is quite high, yet not perfect. This inaccuracy will most

Table 1: Selected parameters with which the network has been trained after fine tuning

Given Parameters	
Parameter	Value
Epochs	50
Batch Size	32
Optimizer	Adam
Learning Rate	10^{-4}
Loss	Binary Cross Entropy
n Feature maps (first layer)	8
Optimizer	Adam

Table 2: Segmentation Results of Manual segmentation per Image.

	Metrics	
Image	DSC	HD
common_40	0.953	3.664
common_41	0.949	5.764
common_42	0.958	3.157

likely have an impact on the atlas based segmentation.

4.2 Atlas based Segmentation

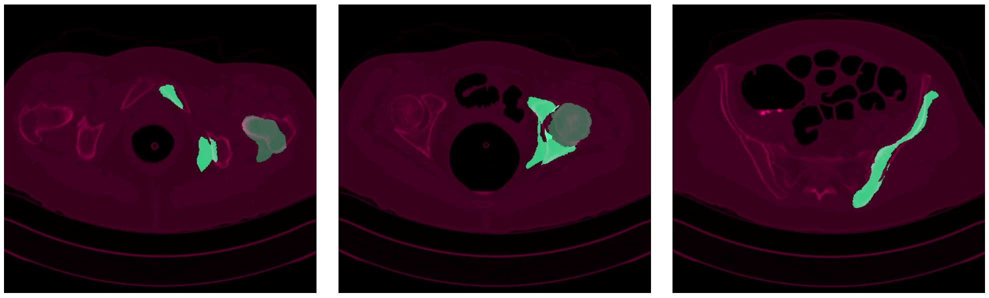
The segmentation results of the atlas-based segmentation are shown in the Table 3. The performances are less accurate than the manual ones and the errors can be due to different steps in the atlas based segmentation process. First of all, the manual segmentation masks that have been used for the majority voting has been realized through our manual segmentation method: the accuracy is not 100%. However, the error is mainly related with the registration process as it is possible to see in the figures below (3 to 5). Here, it can be seen that specifically thin parts seem to be hard to segment. But also in some other parts, there is leakage of segmentation masks into the backgorund class. Some of the segmentation errors could be compensated by adding a proper post-processing, such as smoothing or morphological closing operators. Overall, the segmentation performance is solid, given that the atlas is small and not perfect.
Visualizations of the segmentations with the original image in the background can be seen in Figure 3 for image ”common_40”, Figure 4 for image ”common_41” and Figure 5 for image ”common_42”.

4.3 Classification

The predicted probability of the presence of the Obturatur foramen is shown with respect to each sagital slice in the Figure 6. It is possible to notice that the network detects two areas in each 3D image in which the probability is higher than zero as it happens also in the training data: the are

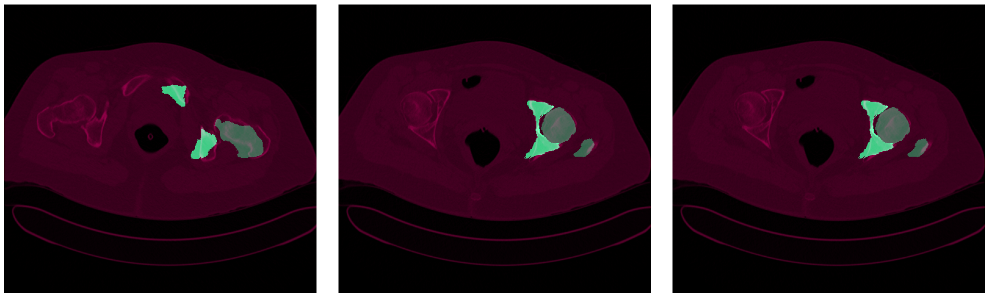
Table 3: Segmentation Results of Atlas based segmentation per Image.

	Metrics	
Image	DSC	HD
common_40	0.749	23.461
common_41	0.752	15.713
common_42	0.850	10.305



(a) (b) (c)

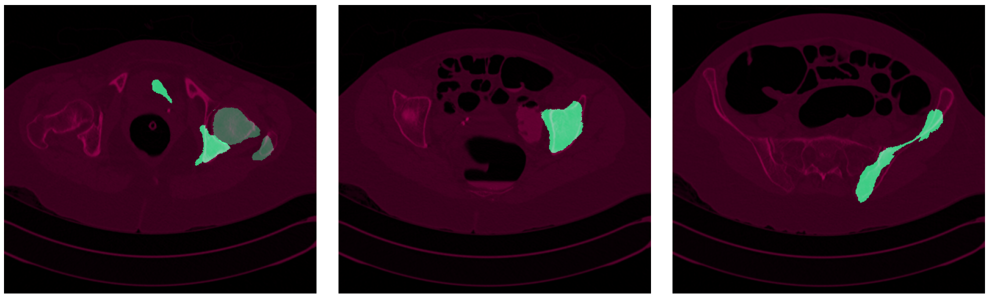
Figure 3: Example slices of the atlas-based segmentation of common_40_image.nii.gz. It can be seen hat the segmentation is quite good, although not perfect. Some thin parts are missing and some segmentation mass is leaking into the background class.



(a) (b) (c)

Figure 4: Example slices of the atlas-based segmentation of common_41_image.nii.gz. It can be seen hat the segmentation is quite good, although not perfect. Some thin parts are missing and some segmentation mass is leaking into the background class.

two groups of adiacent slices that contains the obturator foramen per 3D image. However, as it is possible to see by taking a look at the slices with the highest probability (Figure 7), only for the 41 image the prediction is actually correct. To justify these very low performances with respect to the ones on the training set, we can affirm that the network is overfitted. In fact, the number of images in the training set is very low if compared with the parameters that have to be trained into the network.



(a) (b) (c)

Figure 5: Example slices of the atlas-based segmentation of common_42_image-nii.gz. It can be seen hat the segmentation is quite good, although not perfect. Some thin parts are missing and some segmentation mass is leaking into the background class.

5 Conclusion

In conclusion, the atlas-based registration works well. However, the performance is still subject of further investigation as the segmentation quality can still be improved. The main reasons for the poor quality is the small number of atlas images as well as the registration method that can still be further optimized i.e. with multi-resolution approaches or a more complex linear registration. Furthermore, the segmentation maps should be post-processed to smooth the labels and close holes in the segmentations.

As for the classification of the Obturator foramen, the performance is poor. This might be because of the small number of training samples and the high imbalance between slices that contain the structure and those that do not. Furthermore, the chosen network might be too complex for the problem at hand, as the classification network is overfitted. Therefore, it is suggested to further optimize the classifier. One possible optimization for a more accurate prediction could be the use of a 3D CNN or the use of augmentation during training.

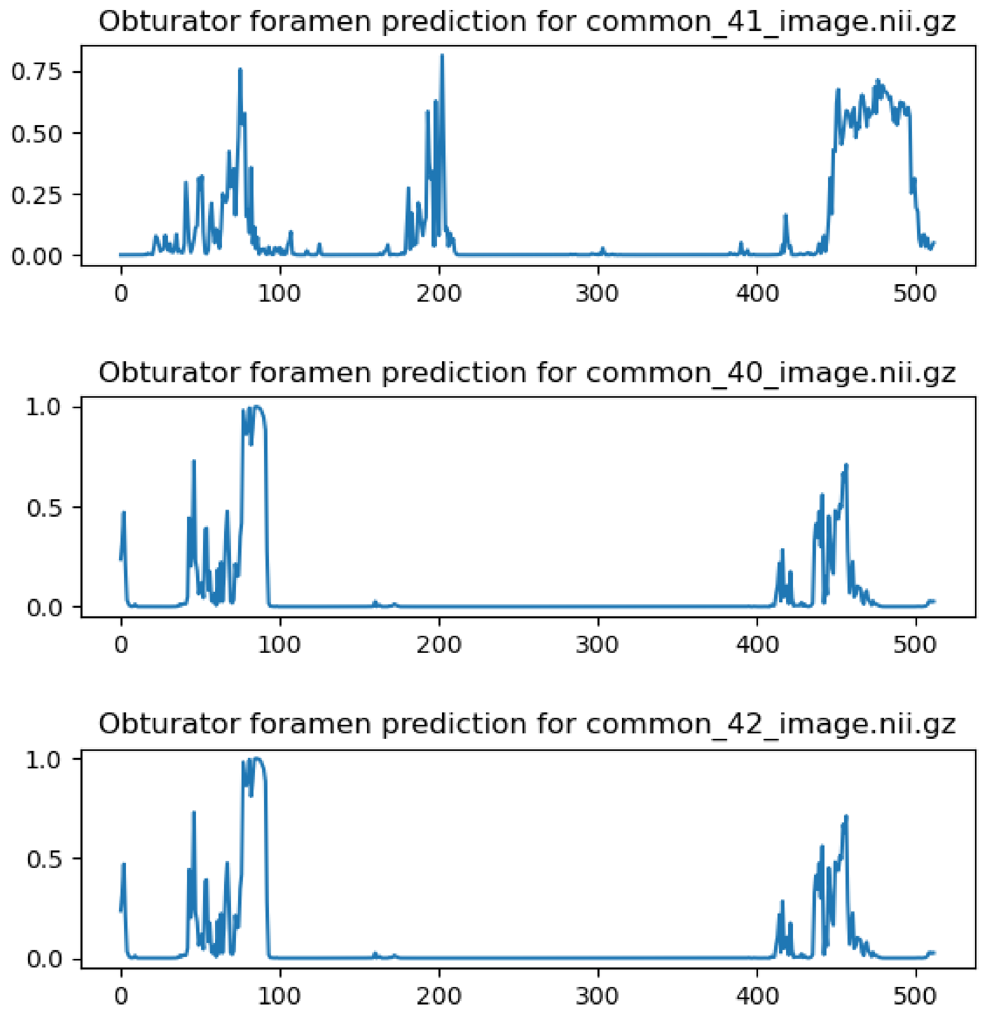


Figure 6: Prediction if Obturatur foramen is present in repsective COMMON image. The x-axis shows the sagital slice number; The y-axis shows the probability of the prediction.

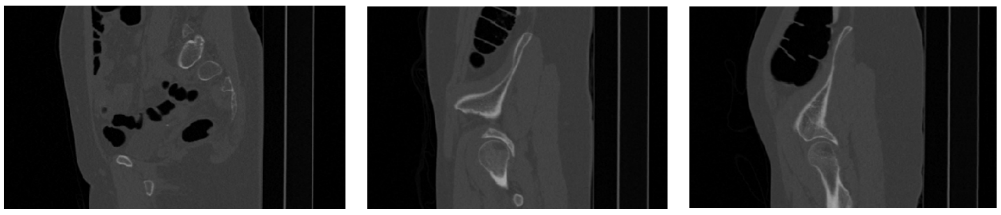


Figure 7: View of the sagital slice that corrisponds to the highest probability prediction for each common image. From left to right: 41 common image slice 202, 40 common image slice 98 and 42 common image slice 98

References

[1] *3D Slicer as an image computing platform for the Quantitative Imaging Network - ScienceDirect*. URL: https://www.sciencedirect.com/science/article/pii/S0730725X12001816 (visited on 12/08/2021).

[2] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

A Code

All code is available on GitHub at https://github.com/lassestahnke/femur_segmentation_atlas.

```python
import SimpleITK as sitk
import os
from assessment import compute_hausdorff, compute_dice
from segmentation import seg_atlas

atlas_masks_base = os.path.join("data", "GROUP_masks")
atlas_images_base = os.path.join("data", "GROUP_images")
save_base_path = os.path.join("data", "atlas_based_segmentations")
im_base_dir = os.path.join("data", "COMMON_images_masks")

files_masks = os.listdir(atlas_masks_base)
files_images = os.listdir(atlas_images_base)
atlas_masks = [os.path.join(atlas_masks_base, fil) for fil in files_masks if ".nii
    " in fil]
atlas_masks.sort()
atlas_images = [os.path.join(atlas_images_base, fil) for fil in files_images if ".
    nii" in fil]
atlas_images.sort()

# load image 40 to segment:
image_40 = sitk.ReadImage(os.path.join(im_base_dir, "common_40_image.nii.gz"),
    sitk.sitkFloat32, imageIO="NiftiImageIO")
gt_seg_40 = sitk.ReadImage(os.path.join(im_base_dir, "common_40_mask.nii.gz"),
    sitk.sitkFloat32, imageIO="NiftiImageIO")

# segment image using the atlas
segmentation_40 = seg_atlas(image_40, atlas_images, atlas_masks)
sitk.WriteImage(segmentation_40, os.path.join(save_base_path, "common_40_image.nii
    .gz"))

# load image 41 to segment:
image_41 = sitk.ReadImage(os.path.join(im_base_dir, "common_41_image.nii.gz"),
    sitk.sitkFloat32, imageIO="NiftiImageIO")
gt_seg_41 = sitk.ReadImage(os.path.join(im_base_dir, "common_41_mask.nii.gz"),
    sitk.sitkFloat32, imageIO="NiftiImageIO")
# segment image
segmentation_41 = seg_atlas(image_41, atlas_images, atlas_masks)
sitk.WriteImage(segmentation_41, os.path.join(save_base_path, "common_41_image.nii
    .gz"))

# load image 42 to segment:
image_42 = sitk.ReadImage(os.path.join(im_base_dir, "common_42_image.nii.gz"),
    sitk.sitkFloat32, imageIO="NiftiImageIO")
gt_seg_42 = sitk.ReadImage(os.path.join(im_base_dir, "common_42_mask.nii.gz"),
    sitk.sitkFloat32, imageIO="NiftiImageIO")
# segment image
segmentation_42 = seg_atlas(image_42, atlas_images, atlas_masks)
sitk.WriteImage(segmentation_42, os.path.join(save_base_path, "common_42_image.nii
    .gz"))
print('finished Registration')

print("start of quality asessment")
# load manually segmented images from Common directory and ground truths
im_base_dir = os.path.join("data", "COMMON_images_masks")
```

```python
manual_seg_base_dir = os.path.join("data", "manual_segmentation")
common40_gt = sitk.ReadImage(os.path.join(im_base_dir, "common_40_mask.nii.gz"),
                             sitk.sitkFloat32, imageIO="NiftiImageIO")
common41_gt = sitk.ReadImage(os.path.join(im_base_dir, "common_41_mask.nii.gz"),
                             sitk.sitkFloat32, imageIO="NiftiImageIO")
common42_gt = sitk.ReadImage(os.path.join(im_base_dir, "common_42_mask.nii.gz"),
                             sitk.sitkFloat32, imageIO="NiftiImageIO")

# load manual segmentations
common40_man = sitk.ReadImage(os.path.join(manual_seg_base_dir, "common_mask", "
    common_40_mask.nii.gz"),
                              sitk.sitkFloat32, imageIO="NiftiImageIO")
common41_man = sitk.ReadImage(os.path.join(manual_seg_base_dir, "common_mask", "
    common_41_mask.nii.gz"),
                              sitk.sitkFloat32, imageIO="NiftiImageIO")
common42_man = sitk.ReadImage(os.path.join(manual_seg_base_dir, "common_mask", "
    common_42_mask.nii.gz"),
                              sitk.sitkFloat32, imageIO="NiftiImageIO")

# comput metrics
common40_dice_man = compute_dice(common40_gt, common40_man)
common40_hd_man = compute_hausdorff(common40_gt, common40_man)
print("Manual Dice common 40: ", common40_dice_man)
print("Manual HD common 40:", common40_hd_man)

common41_dice_man = compute_dice(common41_gt, common41_man)
common41_hd_man = compute_hausdorff(common41_gt, common41_man)
print("Manual Dice common 41: ", common41_dice_man)
print("Manual HD common 41:", common41_hd_man)

common42_dice_man = compute_dice(common42_gt, common42_man)
common42_hd_man = compute_hausdorff(common42_gt, common42_man)
print("Manual Dice common 42: ", common42_dice_man)
print("Manual HD common 42:", common42_hd_man)

# evaluate atlas based method
common40_dice = compute_dice(segmentation_40, common40_gt)
common40_hd = compute_hausdorff(segmentation_40, common40_gt)
print("common 40: dice of segmentation", common40_dice, "HD:", common40_hd)

common41_dice = compute_dice(segmentation_41, common41_gt)
common41_hd = compute_hausdorff(segmentation_41, common41_gt)
print("common 41: dice of segmentation", common41_dice, "HD:", common41_hd)

common42_dice = compute_dice(segmentation_42, common42_gt)
common42_hd = compute_hausdorff(segmentation_42, common42_gt)
print("common 42: dice of segmentation", common42_dice, "HD:", common42_hd)

# optional visualization
# print("Visualizing Segmentation")
# simg1 = sitk.Cast(sitk.RescaleIntensity(image), sitk.sitkUInt8)
# simg2 = sitk.Cast(sitk.RescaleIntensity(segmentation), sitk.sitkUInt8)
# cimg = sitk.Compose(simg1, simg2, simg1 // 2.0 + simg2 // 2.0)
# sitk.Show(cimg, "final segmentation")
```

Listing 1: main.py

```python
import SimpleITK as sitk
from registration import est_lin_transf, est_nl_transf
from transformation import apply_transf


def seg_atlas(im, atlas_ct_list, atlas_seg_list):
    """
    Apply atlas-based segmentation of ‘im‘ using the list of CT images in ‘atlas_ct_list‘ and the corresponding
    segmentation masks in ‘atlas_seg_list‘. Return the resulting segmentation mask after majority voting.
    :param  im: [sitk image] image to segment
            atlas_ct_list: [List(str)] list of paths to images to incorporate into atlas
            atlas_seg_list: [List(str)] list of paths to masks to incorporate into atlas
    :return segmented image
    """

    atlas_segmentations = []  # list to store segmentations

    # register all images in atlas to im and transform their segmentation masks
    print(len(atlas_ct_list), "atlas images found")

    for i in range(len(atlas_ct_list)):
        print("Register image", i + 1, " out of ", len(atlas_ct_list))
        # read image and mask
        im_atlas = sitk.ReadImage(atlas_ct_list[i], sitk.sitkFloat32, imageIO="NiftiImageIO")
        mask_atlas = sitk.ReadImage(atlas_seg_list[i], sitk.sitkUInt16, imageIO="NiftiImageIO")

        # estimate and apply linear transform
        print("Register image", i + 1, " out of ", len(atlas_ct_list), "... Estimate linear registration")
        transf_lin = est_lin_transf(im, im_atlas)
        im_atlas_lin = apply_transf(im, im_atlas, transf_lin)

        # estimate non-linear transform
        print("Register image", i + 1, " out of ", len(atlas_ct_list), "... Estimate non-linear Registration")
        transf_nl = est_nl_transf(im, im_atlas_lin)

        # apply linear and non-linear transformation to atlas mask
        segmentation = apply_transf(im, mask_atlas, transf_lin)
        segmentation = apply_transf(im, segmentation, transf_nl)
        segmentation = sitk.Cast(segmentation, sitk.sitkUInt16)

        # add to atlas segmenation for majority voting
        atlas_segmentations.append(segmentation)

    return sitk.LabelVoting(atlas_segmentations)
```

Listing 2: segmentation.py

```python
"""
This file contains all methods used for estimating the transform for registration of a moving image to a reference
image. This file was written by https://github.com/lassestahnke and https://github.com/RebeccaBonato with the help of
sitk tutorials.
```

```python
"""

import SimpleITK as sitk


def command_iteration(method):
    """
    Function to print current number of iterations and current metric value.
    :param method:
    :return:
    """
    print(
        f"{method.GetOptimizerIteration():3} "
        + f"= {method.GetMetricValue():10.5f} "
        + f": {method.GetOptimizerPosition()}"
    )


def est_lin_transf(im_ref, im_mov):
    """
    Estimate linear transform to align `im_mov` to `im_ref` and return the transform parameters.
    :param  im_ref: [sitk image] reference image
            im_mov: [sitk image] moving image
    return: transformation
    """
    im_ref_mask = im_ref > 150
    im_ref_mask = sitk.Cast(im_ref_mask, sitk.sitkInt16)

    im_mov_mask = im_mov > 150  # only look at strong signals
    im_mov_mask = sitk.Cast(im_mov_mask, sitk.sitkInt16)

    initial_transform = sitk.CenteredTransformInitializer(im_ref_mask,
                                                          im_mov_mask,
                                                          sitk.Similarity3DTransform(),
                                                          sitk.CenteredTransformInitializerFilter.MOMENTS
                                                          )

    # Set methods for registration; Start with Linear
    R = sitk.ImageRegistrationMethod()
    R.SetMetricAsMattesMutualInformation()
    R.SetOptimizerAsRegularStepGradientDescent(1.2, 0.01, 20)
    R.SetOptimizerScalesFromPhysicalShift()
    R.SetMetricSamplingStrategy(R.NONE)
    R.SetMetricSamplingPercentage(0.10)
    R.SetInitialTransform(initial_transform, inPlace=False)
    R.SetInterpolator(sitk.sitkLinear)
    R.AddCommand(sitk.sitkIterationEvent, lambda: command_iteration(R))

    return R.Execute(im_ref, im_mov)


def est_nl_transf(im_ref, im_mov):
    """
    Estimate non-linear transform to align `im_mov` to `im_ref` and return the transform parameters.
    :param  im_ref: [sitk image] reference image
```

```python
            im_mov: [sitk image] moving image
    return: transformation
    """

    im_ref_mask = im_ref > 150
    im_ref = sitk.Normalize(im_ref)
    im_ref = sitk.DiscreteGaussian(im_ref, 3)

    im_mov_mask = im_mov > 150  # only look at strong signals
    im_mov = sitk.Normalize(im_mov)
    im_mov = sitk.DiscreteGaussian(im_mov, 3)

    transformDomainMeshSize = [8] * im_mov.GetDimension()
    tx = sitk.BSplineTransformInitializer(im_ref, transformDomainMeshSize)

    print("Initial Parameters:")
    print(tx.GetParameters())

    R = sitk.ImageRegistrationMethod()
    R.SetMetricAsMattesMutualInformation(50)
    R.SetOptimizerAsGradientDescentLineSearch(
        5.0, 100, convergenceMinimumValue=1e-4, convergenceWindowSize=5
    )
    R.SetMetricSamplingStrategy(R.RANDOM)
    R.SetMetricSamplingPercentage(0.01)
    R.SetMetricMovingMask(im_mov_mask)
    R.SetMetricFixedMask(im_ref_mask)

    R.SetInitialTransform(tx, inPlace=False)
    R.SetInterpolator(sitk.sitkLinear)
    # R.AddCommand(sitk.sitkIterationEvent, lambda: command_iteration(R))

    return R.Execute(im_ref, im_mov)
```

Listing 3: registration.py

```python
import SimpleITK as sitk


def apply_transf(im_ref, im_mov, xfm):
    """
    Apply given transform `xfm` to `im_mov` and return the transformed image.
    """
    resampler = sitk.ResampleImageFilter()
    resampler.SetReferenceImage(im_ref)  # reference image for size, origin and
    spacing
    resampler.SetInterpolator(sitk.sitkNearestNeighbor)
    resampler.SetDefaultPixelValue(0)
    resampler.SetTransform(xfm)
    return resampler.Execute(im_mov)
```

Listing 4: transformation.py