1 DICOM Loading and Visualization

The CT image is composed of 43 DICOM files, with each one representing a slice of the full 3D CT image. Since we want to work with a single object, a 3-dimensional array in this case, we need to arrange these slices to build our complete CT image. To accomplish this, we first need to inspect the headers. We are using the PyDICOM package to read and access DICOM files.

1.1 Header Inspection

DICOM headers contain relevant metadata of both the CT image and the patient: demographic information, image type, slice thickness, position of the patient, etc. Here are the ones used or relevant for our task:

- **SliceLocation**: this attribute refers to the relative position of the image plane expressed in mm. As a very straightforward solution, we can use this parameter to sort the slices (each DICOM file) and them stacking them to build the full 3D CT image file. Equivalently, we could use the third coordinate of **Image Position Patient**.
- **SliceThickness**: as implied by its name, it indicates the thickness in mm of the slice. If all the slice thicknesses of all the slices were the same, we could think of it as the conversion from pixels to mm of the z coordinate. We will use it to adjust the aspect ratio when plotting the CT image.
- **Acquisition number**: is a number identifying the single continuous gathering of data over a period of time that resulted in this instance. So we can this parameter to ensure that all the DICOM files come from the same series, and indeed, in our case they come, all files have the same acquisition number.
- **ReferencedSegmentationNumber**: carries the identification of the segment a slice it belongs to. If the segmentation slice represents a Liver mask slice with ID 1, this value will be 1.

1.2 CT image rearrangement

Having already inspected the headers, we know how can we rearrange the DICOM files. The idea is to read all the files of the series folder, and append them into a list, obtaning for each one its slice location. After that we can now sort this list using the slice location as a sorting criteria. Finally, we can stack all the slices, now ordered from the bottom to the top, thus constructing the 3D CT image.

1.3 Artifact removal

While inspecting the CT image with the 3D slicer software, we noticed that there was an artifact behind the patient's body. We think that perhaps could be the hosptial stretcher where the patient was on. This object could diminish the correct visibility of the CT image and segmentation in the GIF animation. Consequently, we aim to remove it.

First we average (in the vertical z axis) all the slices into one, ignoring the negative values to avoid perturbing the mean. At this point, we'll have two clearly defined blobs: the body and the stretcher. Then, we binarize the image and find the largest contour. Finally, this contour will serve as a mask to apply to every slice of the CT image, thus eliminating the artifact throughout the image.

1.4 Rotating MIP

1.4.1 Segmentation Image

Our segmentation image comes as a single file, so there's no need for rearrangement to build the segmentation. It contains 4 classes: Liver, Mass (tumor), Portal Vein, and abdominal aorta. Where it really needs rearrangement is when we must align this segmentation with the CT image it corresponds to. To achieve this, we will create a blank canvas with the size of the CT image and then assign each slice of the segmentation according to the SliceLocation that best corresponds with the third coordinate of the ImagePositionPatient of the segmentation image. In each slice, instead of having a binary value of 0 or 1, the pixels will have the ReferencedSegmentNumber of that slice, thus representing the class (Liver, Mass, etc). This approach ensures the creation of a segmentation that accurately matches the CT image.

1.4.2 Alpha Fusion

Alpha fusion refers to a technique where multiple images are combined using a weighted average. The method aims to enhance the visual or diagnostic quality of the resulting image by emphasizing important features. By adjusting the alpha parameter, we can control the degree of influence each image has on the final result. In our 2-image case we will just apply the following equation:

$$\text{Fusion}_{\text{img}} = \text{CT}_{\text{img}} \cdot (1-\alpha) + \text{Segmentation}_{\text{img}} \cdot \alpha \tag{1}$$

But prior to that, we first need to apply a colormap to both images, to convert scalar values to colors. We will use `bone` for mapping the CT image, this is a sequential grayscale colormap with a tinge of blue. We use this colormap in order not to highlight any specific parts of the body but at the same time allowing them to differentiate from one another. And for the segmentation image we are going to apply a qualitative colormap named `Set1`, we decided to use this colormap to emphasize each of the different segmentation classes by assigning them a color with a notable difference in hue to enable easy differentiation between them using distinct colors. It is fair to mention that before applying a colormap we've had to apply a min-max normalization in order to avoid binarization of the images when applying the map.

1.4.3 GIF Creation

Now lets explain how we created a rotating animation of the CT and segmentation fused. Firstly, we needed to adjust both the CT and segmentation images to the same size to combine them using alpha fusion. Therefore, we cropped the CT image, which was the largest, to match the size of the segmentation image.

Then, within a loop, since we have to retrieve the projection on the sagittal (or coronal) plane, in each iteration we applied a rotation of the fused image by a certain angle (determined by the loop). After that, we applied Maximum Intensity Projection on the sagittal plane and stored it. Later on, after completing all iterations, we can generate the GIF animation using the capabilities of matplotlib. When creating animation frames, it's important to readjust the aspect ratio based on the mentioned slice thickness. The vertical pixel distances in the axial axis represent five times more millimeters than those of the other axes. The Maximum Intensity Projection on the sagittal plane was build by just computing the maximum values on the third axis. We can see a frame of the final result in the figure 1

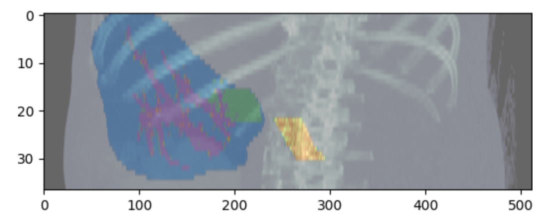


Figure 1: Frame of the resulting GIF.

2 3D Rigid Coregistration

2.1 Loading and transforming DICOM files

Firstly we load and rearrange the patient's brain image by using the same method as the previous section. But now, retrieving not only the slice thickness, but also the pixel spacing. Then we load our reference image, that is made by averaging 152 brain images, from now on we are going to name this image the *Phantom Image* due to its appearance.

The next step is to resize the input image so that both the input and phantom images have the same physical proportions in mm^3. For this purpose, we will use the Pixel Spacing parameter. This attribute represents the actual distance in millimeters between the center of a pixel and its neighbour. It consists of two values: one for column pixel spacing and another for row pixel spacing. Thus, since the PixelSpacing of the input image is ≈ 0.5 (for both rows and columns) and for the phantom image is 1 (for both rows and columns), we will apply a scaling factor to the axes representing the rows and columns. This scaling factor will be the division of the pixel spacing of the image we want to resize by the pixel spacing of the target image we want to resemble.

Now, we will notice that, although the shapes do not match, the brains more or less have the same width and heights measured in pixels. Finally, we will only need to correct these slight changes in shape by cropping the largest image to match the dimensions of the smallest image. In addition to that, since the Phantom and input images have different range values (0-90) (0-515) respectively, we will need to apply a normalization, in our case a min-max. This ensures that we work with consistent sizes and ranges during coregistration, fact that will be very useful for computing our loss function easily.

2.2 Image Rigid Coregistration

Rigid coregitration refers to aligning two or more images by applying transformations that preserve distances and angles, such as translation and rotation. The purpose here is to match the input patient image to the reference image in order to locate and highlight the Thalamus region into the patient's brain.

2.2.1 Optimization Function

The main idea is to implement an optimization approach, in which we attempt to minimize a function that applies a rigid transformation, such as translation followed by axial rotation, and then it compares the result with the reference image. Subsequently, we compare our input image with the reference image by computing a loss function, MSE in our case, between the reference and input images. In this way, step by step, we transform the input image to increasingly resemble (spatially) the reference image with each iteration, leading to a coregistration of both images. We experimented with various optimization algorithms from the Python package `scipy.minimize`, and the one that brought the best results was the Nelder-Mead method, with an optimal MSE of 0.0580 (our images are normalized to [0-1]). You can find the optimal MSE and execution time for the other methods we attempted in Table 1.

Method	L-BFGS-B	BFGS	**Nelder-Mead**	Powell	CG	SLSQP	COBYLA	TNC
MSE	0.064	0.064	**0.058**	0.11	0.064	0.114	0.061	0.063
Time (s)	147	798	332	302	575	28	141	97

Table 1: MSE and execution time of the coregistration using different optimization methods.

2.2.2 Initial parameters

We noticed that the reference brain and input brain were reversed in the first and second axis by an angle of 2π. Therefore when we firstly applied the optimization methods, the algorithm was probably trapped in a local minima where every change lead to worse loss function results. Thus, the resulting optimal parameters were the same, or almost the same, as the initial parameters.
To change this behavior, we adjusted the initial parameters so that both brains started with the same spatial orientation. First building a rotation matrix that encompassed all the transformations into one matrix: $R_t = T_1 \cdot R_x(2\pi) \cdot T_2 \cdot R_y(2\pi)$, and then converting that rotation matrix into a translation and axial rotation parameters $(t1, t2, t3, \theta, v1, v2, v3)$. You can see the full procedure in the GitHub repository.

2.2.3 Inverse Transformations Approach

We sadly cannot directly compute those transformations into the pixel values of the images. Instead, we compute the transformations on the pixel coordinates (x, y, z), and then evaluate the coregistration (MSE) using the values in those coordinates. Unfortunately, if we apply the transformations to the coordinates from the input image, some pixels will not have a representation in the transformed canvas. Thus some strange patterns will appear in the transformed image, as you can see in figure 2. To avoid this, we will reverse our approach. We will apply the inverse transformation to every (x_c, y_c, z_c) coordinate on the transformed (not yet) canvas. Instead of using translation and then axial rotations with parameters: $(t1, t2, t2)$ for translation and (θ,v1,v2,v3) for axial rotation; what will do is first an axial rotation with parameters (θ,-v1,-v2,-v3) and then a translation $(-t1, -t2, -t3)$. In this way, we can determine the corresponding value for that specific pixel of the canvas in the transformed space. We can represent the transformed image and the MSE as follows:

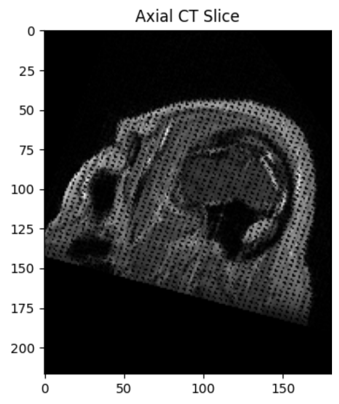


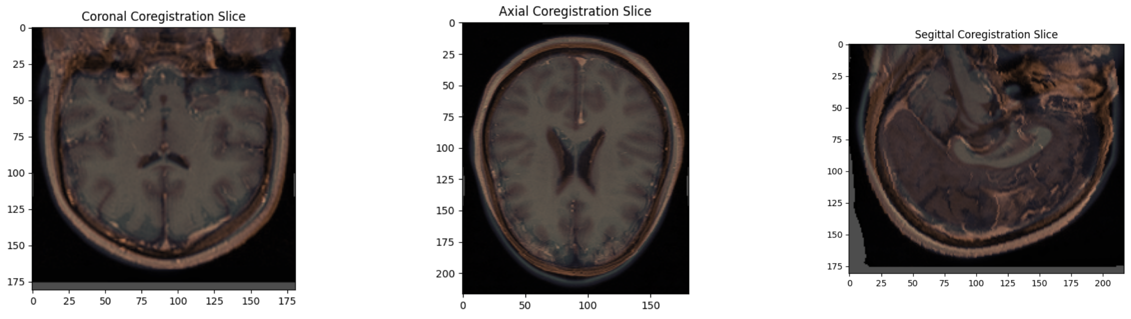
Figure 2: Pattern appareance due to unrepresented pixels in normal transformation approach.

$$\text{Trans}_{\text{IMG}}[x_c, y_c, z_c] = \text{Input}_{\text{IMG}}[t^{-1}(x_c, y_c, z_c)] \tag{2}$$

$$\text{MSE} = \frac{1}{w \cdot h \cdot d} \sum (\text{Trans}_{\text{IMG}} - \text{Ref}_{\text{IMG}})^2 \tag{3}$$

Additionally, it's important to consider that many pixel coordinate transformations will result in coordinates falling outside the canvas. In such cases, those pixels are simply not painted on the canvas.

Because we need to apply the transformation to millions of pixel coordinates, we will use the Python package `quaternion`. This package allows us to use NumPy arrays with a *dtype* of quaternion objects, and it also implements operations such as multiplication, division, addition, subtraction, and conjugation for quaternions. This approach involves converting the coordinates to quaternions and then applying the transformation using quaternion multiplication by vectorized operations, which is significantly faster than executing a Python loop, you can see the code in the github repository form above. To verify visually the success of the coregistration we extracted the middle planes (axial, coronal, and sagittal) slices, you can see them in figure 3.


Coronal Coregistration Slice
Axial Coregistration Slice
Segittal Coregistration Slice
Figure 3: Middle slice planes. Coronal, axial, and sagittal respectively; of the coregistrated brains, the reference brain is in white and input brain in *copper*.

2.3 Visualize Thalamus region on patient's input space

Now, the final step is to visualize the Thalamus region in the patient's input space. To accomplish this, we will extract the pixels corresponding to the thalamus region from the atlas image with the text file information, creating a mask. Then, we'll transform all of those pixels from the reference space to the input space. Finally, we'll visualize the thalamus region along with the patient's brain using alpha fusion.

2.3.1 Obtaining Mask

The atlas is a 3D image in the reference space where each pixel value is an ID representing the class (region) it belongs to: hippocampus, cerebellum, thalamus, etc. First, we needed to identify which pixel IDs were representing the entire Thalamus region. To achieve this, we inspected the `AAL3_1mm.txt` file and found all the IDs related to the Thalamus region. Finally, with all the pixels that had the same value of one of those IDs, we created a mask of the Thalamus region in the reference space.

2.3.2 Inverse transformation

Now, we needed to pass all those pixels from the mask into the patient's input space. To accomplish this, we apply the inverse transformation obtained through the optimization method. This involves applying an axial rotation followed by a translation. In this way, we go from the reference space to the input space. This is not directly straightforward though, because in order to avoid the appearance of black patterns due to the unrepresented pixels in the transformed canvas, as we said in the previous section, we need an inverse transformation approach, in which the transformation is already inverted. Thus, we apply the inverse transformation approach but without inverting the transformation nor the parameters. Consequently, we use the inverse approach but applying a translation and then an axial rotation with the exact optimal parameters obtained in the optimization part.

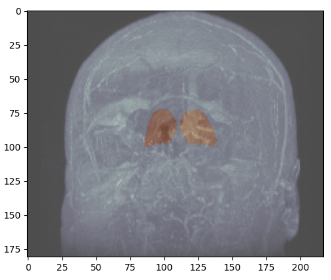


Figure 4: Thalamus visualization on the patient's input space.

2.3.3 Mask visualization

Now, to visualize (and highlight) the thalamus region in the patient's input brain, we will apply a colormap to both and then merge them using alpha fusion. For the thalamus, we used the `tab10` qualitative colormap to obtain a single color, for highlighting the thalamus; while for the patient's input brain, we used the `bone` colormap to distinguish between different parts of the body without highlighting anyone. You can see the result in figure 4

3 Findings and shortcomings

4 Findings

One notable finding is the application of the inverse transformation approach in the rigid coregistration part to avoid the appearance of black patterns from unrepresented pixels. In this way we omit the need of interpolation, which would lead to a worse performance. This method ensures better algorithm performance and allows for the representation of all necessary pixels on the transformed canvas without additional processing steps.
The comparison of optimization methods for rigid coregistration revealed differences in performance (regarding MSE), with the Nelder-Mead algorithm showing a better accuracy. Establishing the most suitable optimization approach could be beneficial in project regarding medical images for diagnostic and research purposes.
This project highlights the collaboration of medical imaging and computational techniques, such as involving clinicians or biologists with computer scientists, combining domain-specific knowledge for the sake of enhancing our health systems.

4.1 Shortcomings

There are several aspects that depend on the specific image and are hardcoded, such as artifact removal, cropping, image resizing, and initial parameters. This lack of adaptability makes the project less suitable for generic use and requires adjustments to accommodate different images.
The optimization time is unacceptable for real-time applications. Optimization iterations may need to be significantly accelerated or alternative approaches explored to meet real-time processing requirements.
The visualization of the thalamus region in the input patient's brain image is blocked by surrounding body tissues such as flesh, bones, and skin. A desirable enhancement would be to selectively display the thalamus region within the patient's brain, isolating it from surrounding , and non-relevant body parts.