Extension de recalage pour 3D Slicer

Wissam Boussella, Iantsa Provost, Bastien Soucasse et Tony Wolff

Un devoir présenté dans le cadre du
projet de fin d'études.

Master informatique pour l'image et le son
Université de Bordeaux, France

Table des matières

1 Introduction **3**

2 Travaux connexes **4**
2.1 3D Slicer . . . 4
2.2 ITK . . . 4
2.3 Elastix . . . 4
2.4 ANTS . . . 5
2.5 Autres bibliothèques . . . 5

3 User Stories **6**
3.1 Intitulés des User Stories . . . 6
3.2 Critères d'acceptabilité . . . 6

4 Tests **9**
4.1 Monkey Testing . . . 9
4.2 Unit Testing . . . 9
4.2.1 Prétraitements . . . 9
4.2.2 Recalage . . . 11
4.2.3 Carte de différence . . . 14

5 Implémentation **15**
5.1 Implémentation globale . . . 15
5.1.1 Module d'extension . . . 15
5.1.2 Architecture générale . . . 16
5.1.3 Architecture des fonctionnalités . . . 16
5.1.4 Images d'entrée/cible et visualisation . . . 17
5.2 Prétraitements . . . 20
5.2.1 Sélection de ROI . . . 20
5.2.2 Rognage . . . 23
5.2.3 Rééchantillonnage . . . 26
5.3 Recalage . . . 27
5.3.1 Implémentation logicielle . . . 27
5.3.2 Limites . . . 31
5.4 Carte de différence . . . 32
5.4.1 Valeur absolue de la différence . . . 32
5.4.2 Différence de gradient . . . 33
5.4.3 Filtre moyenneur avec noyau de convolution . . . 33
5.4.4 Look Up Table . . . 33
5.5 Plugins . . . 34

6 Conclusion **37**
6.1 Bilan . . . 37
6.2 Perspectives . . . 37
6.3 Notes de fin . . . 38

7 Annexes **40**

1 Introduction

Dans le contexte de notre projet de fin d'études, nous avons développé un outil de recalage d'images 3D dans le domaine médical. Il s'agit d'une extension pour le logiciel de traitements d'images 3D médicales 3D Slicer.

La volonté du client était d'agréger en une même interface au sein de 3D Slicer des algorithmes de prétraitements (en particulier sélection de ROI, rognage, et rééchantillonnage), de recalage, et de calcul de carte de différence, tout en donnant la possibilité d'ajouter le plus simplement possible de nouveaux algorithmes a posteriori. Dans le cadre d'une extension intégrée à 3D Slicer, le client avait également la volonté de pouvoir visualiser les images (avant et après traitement) et les cartes de différences en parallèle.

Le projet a donc été développé en Python de sorte de correspondre à 3D Slicer. Différentes bibliothèques sont intégrées à 3D Slicer et sont à utiliser dans ce projet. Le client nous a informé vouloir se contenter d'algorithmes de deux bibliothèques dont il connaît l'utilisation : ITK et Elastix.

Le projet a été conçu et mené en suivant la méthode agile SCRUM, c'est-à-dire que le projet a été développé sur différents *sprints* (succession de tâches à terminer avant la prochaine itération) et axé sur les tests avant l'implémentation : c'est le Test-Driven Development (TDD).

Ce rapport synthétise notre expérience sur le projet. Il décrit tout d'abord les travaux connexes au traitement et recalage d'images 3D médicales. Nous explicitons ensuite la conception du projet en détaillant les User Stories qui représentent très précisément les différents besoins du client, qui nous ont aidé à garder une ligne directrice durant le développement du projet. Les tests sont ensuite décrits dans ce rapport, avant de détailler concrètement l'implémentation. Celle-ci est complète et agrémentée d'illustrations et captures d'écran pour une description des plus précise. Nous concluons enfin avec notre retour sur le projet, la réponse aux besoins du client, et les perspectives futures envisagées par l'équipe.

FIGURE 1 – Custom Registration Project

2 Travaux connexes

En amont de la conception de notre projet, nous avons recherché les différentes bibliothèques et les différents logiciel dédiés aux traitements d'images 3D dans le domaine médical, en particulier au recalage d'images.

2.1 3D Slicer

3D Slicer, introduit par Fedorov et al. (2012), est un logiciel de traitement d'images médicales en 3D. Bien que le logiciel offre des fonctionnalités de prétraitement et de recalage, celles-ci sont réparties dans différents modules du logiciel, ce qui peut rendre son utilisation complexe. De plus, les algorithmes sous-jacents sont souvent complexes et requièrent de nombreux paramètres avancés à renseigner. En conséquence, l'utilisation de ce logiciel peut prendre du temps et être difficile.

En outre, il est à noter que le logiciel ne propose pas de fonctionnalité pour calculer des cartes de différences entre différentes versions d'images. De plus, l'ajout de nouvelles fonctionnalités ou algorithmes de traitement peut s'avérer fastidieux, le logiciel n'étant pas prévu pour l'utilisation de scripts personnels.

Cette complexité peut donc limiter l'adaptabilité et la flexibilité du logiciel dans le contexte de l'imagerie médicale, où les besoins et les exigences sont en constante évolution.

2.2 ITK

ITK est une bibliothèque C++ de traitement d'images médicales, introduite par Yoo et al. (2002). La bibliothèque offre à la fois des fonctionnalités de prétraitement et de recalage d'images médicales en 3D, et utilisation est possible grâce à son interface Python.

ITK étant une bibliothèque, il n'y a pas d'interface graphique native. Le logiciel 3D Slicer fait appel aux algorithmes de cette bibliothèque dans son implémentation, ce qui permet un affichage interactif et une manipulation plus aisée des algorithmes.

SimpleITK (Lowekamp et al., 2013) est une sur-couche pour ITK, une interface Python qui offre un accès simplifié aux algorithmes de la bibliothèque, au détriment de certaines fonctionnalités avancées.

2.3 Elastix

Elastix est une bibliothèque C++ basée sur ITK, introduite par Klein et al. (2010). Il s'agit d'un regroupement d'outils pour le recalage rigide et non-rigide d'images médicales en utilisant des méthodes élastiques. Elle comprend également une interface Python.

De la même manière que pour ITK, Elastix n'a pas d'interface graphique native, s'agissant d'une bibliothèque. 3D Slicer possède néanmoins une extension (non installée par défaut) pour intégrer les algorithmes d'Elastix au logiciel, en fournissant une interface graphique,ainsi qu'une interaction en temps réel.

SimpleElastix est une sur-couche simplifiée pour Elastix, proposée par Marstal et al. (2016). Elle permet une utilisation plus facile et plus intuitive d'Elastix, au même titre que SimpleITK pour ITK.

2.4 ANTS

ANTS (Advanced Normalization Tools) est une bibliothèque de recalage d'images médicales en 2D et 3D, développée par Avants et al. (2008). Elle utilise une méthode d'optimisation basée sur les déformations élastiques pour effectuer le recalage, et offre des fonctionnalités avancées telles que l'inclusion de données anatomiques a priori pour guider le recalage. ANTS possède également une interface graphique.

Cependant, ANTS est un logiciel relativement complexe à utiliser de part ses nombreux paramètres distincts qui peuvent rendre la configuration pour un recalage optimal difficile. ANTS est également connu comme sensible aux artefacts d'image, ce qui est peu approprié aux diverses méthodes d'acquisition. Enfin, les temps de calcul sont généralement très longs.

2.5 Autres bibliothèques

Il existe d'autres bibliothèques et logiciels spécialisés dans le recalage d'images 3D dans le domaine médical. Cependant, la plupart de ces outils sont limités à des images cérébrales et/ou des images acquises par IRM. On peut notamment citer BRAINS (Brain Research : Analysis of Images, Networks, and Systems), SPM (Statistical Parametric Mapping).

3 User Stories

Dans le cadre de ce projet, nous avons établi un ensemble de User Stories (US) qui encapsulent les fonctionnalités principales de l'application logicielle. Celle-ci propose de manière interactive des prétraitements, des algorithmes classiques de recalage d'images 3D ainsi que le calcul de cartes de différence. Pour une utilisation avancée, cette application devra être extensible grâce à la possibilité d'intégrer des plugins personnalisés.

Ces US décrivent les fonctionnalités que le logiciel doit prendre en charge pour répondre efficacement aux besoins des utilisateurs finaux.

Deux types d'acteurs sont désignés dans les US : les utilisateurs et les développeurs. Les utilisateurs seront les principaux utilisateurs finaux du logiciel, et auront besoin d'une interface minimale pour effectuer les traitements sur les images 3D. Les développeurs, quant à eux, seront responsables de l'implémentation des plugins, et devront être en mesure d'importer des interfaces XML et des scripts Python pour ajouter des fonctionnalités supplémentaires.

3.1 Intitulés des User Stories

La première US, axée sur les prétraitements d'images, est formulée de la manière suivante : **En tant qu'utilisateur, je dois pouvoir appliquer des prétraitements (sélection de ROI par seuillage, rognage manuel ou automatique et rééchantillonnage automatique) avec des paramètres minimaux, à une image d'entrée, en fonction d'une image de référence, via une interface dédiée les regroupant, en affichant les deux images ainsi qu'une prévisualisation en temps réel (I)**.

La deuxième US s'intéresse quant à elle à l'application d'algorithmes de recalage, comme suit : **En tant qu'utilisateur, je dois pouvoir appliquer des algorithmes de recalage rigide ou non-rigide, paramétrables manuellement ou à l'aide de presets, via une interface dédiée, en affichant l'image à recaler et l'image de référence, puis l'image recalée, dans la vue (II)**.

La troisième US portant sur la génération de carte de différence se présente sous la forme suivante : **En tant qu'utilisateur, je dois pouvoir calculer une carte de différence entre deux images choisies, voxel-à-voxel ou par gradients et avec possibilité d'utiliser des patchs, via une interface dédiée, et de l'afficher dans la vue (III)**.

La dernière US concerne l'intégration de plugins. Nous l'avons exprimé de la façon suivante : **En tant que développeur, je dois pouvoir importer une interface XML et un script Python à utiliser dans le cadre d'un plugin (IV)**.

3.2 Critères d'acceptabilité

Pour chaque US, nous avons établi une liste de critères d'acceptabilité. L'ensemble des critères relatifs à une US doit être satisfait pour que celle-ci soit considérée comme accomplie avec succès. Également, afin d'illustrer ce qui est attendu pour l'UI, nous avons esquissé une panoplie de maquettes.

Avant tout, la maquette générale (Figure 2) a été conceptualisée en se basant sur le design de 3D Slicer avec deux parties : la partie gauche dédiée au panneau de traitements où on retrouve toutes nos fonctionnalités, la partie droite dédiée à la visualisation.

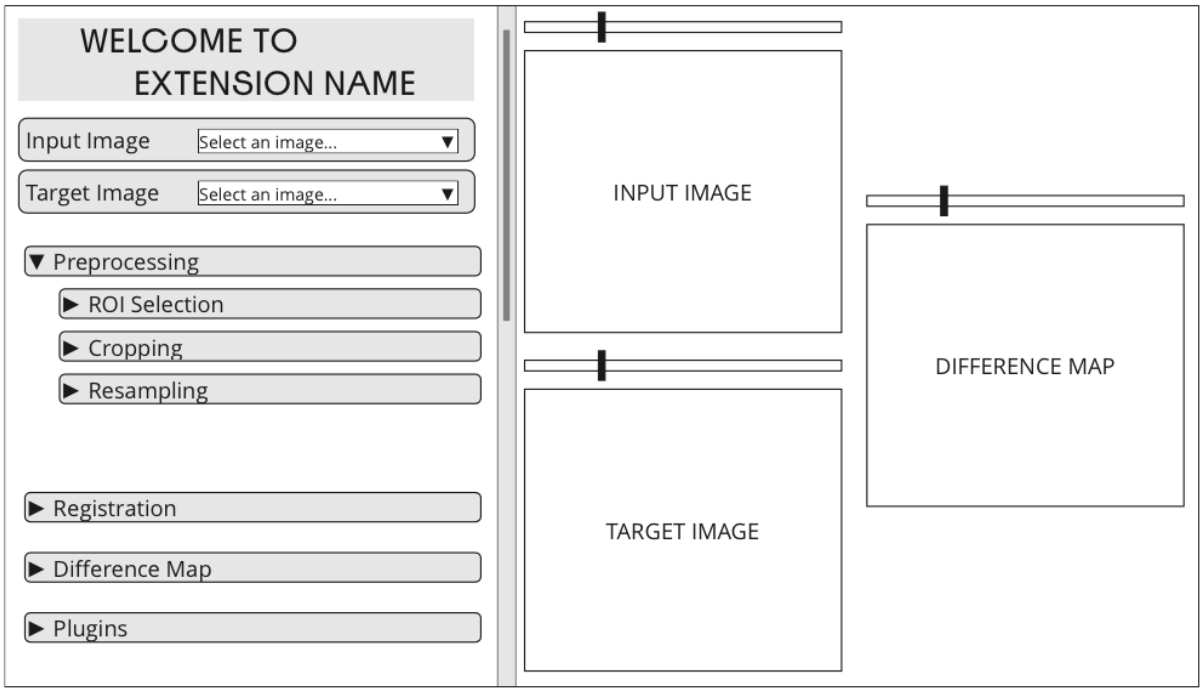

FIGURE 2 – Maquette de 3D Slicer ouvert sur notre module d'extension.

Pour la US I, l'interface de prétraitements doit permettre de sélectionner l'image d'entrée, et celle de référence lorsque c'est nécessaire (blocs de sélection *Input Image* et *Target Image*, Figure 2) et de les afficher, tout comme l'image résultante (blocs d'affichage *Input Image* et *Target Image*, Figure 2). Ensuite, les options de prétraitement doivent inclure la sélection de ROI par seuillage, le rognage manuel et automatique, ainsi que le rééchantillonnage automatique (bouton dépliant *Preprocessing*, Figure 2). Elles doivent également être paramétrables via l'interface dédiée (Figures 3, 4 et 5). Enfin, l'interface doit afficher une prévisualisation du traitement souhaité, en temps réel.

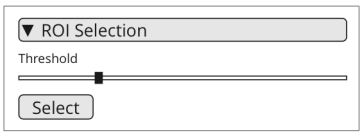

FIGURE 3 – Maquette de l'UI de sélection de ROI.

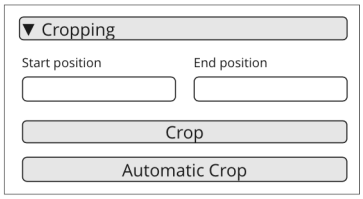

FIGURE 4 – Maquette de l'UI de rognage.

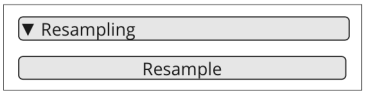

FIGURE 5 – Maquette de l'UI de rééchantillonnage.

En ce qui concerne la US II, l'interface de recalage doit non seulement permettre de sélectionner l'image à recaler et celle de référence (blocs de sélection *Input Image* et *Target Image*, Figure 2), mais elle doit aussi les afficher tout comme l'image recalée générée (blocs d'affichage *Input Image*, *Target Image* et *Difference Map*, Figure 2). Quant aux options de recalage, elles doivent inclure des algorithmes rigides et non rigides paramétrables manuellement ou à l'aide de presets (Figure 6).

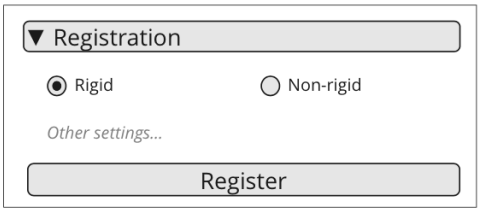

FIGURE 6 – Maquette de l'UI de recalage.

Quant à la US III, l'interface de calcul de la carte de différence doit permettre de sélectionner les images à comparer (blocs de sélection *Input Image* et *Target Image*, Figure 2) mais aussi d'afficher la carte calculée dans la vue (bloc d'affichage *Difference Map*, Figure 2). En outre, les options de calcul de la carte de différence doivent inclure la possibilité de calculer la différence voxel-à-voxel ou par gradients, et de spécifier l'utilisation de patchs (Figure 7).

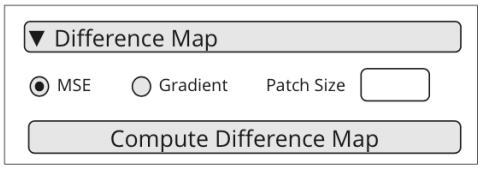

FIGURE 7 – Maquette de l'UI de génération de carte de différence.

Pour finir, pour la US IV, l'interface de développement doit permettre l'importation d'une interface XML et d'un script Python pour une utilisation en tant que plugin.

Ensuite, ces plugins doivent être intégrables dans le logiciel principal et accessibles via une interface dédiée (Figure 8). Enfin, les plugins doivent être compatibles avec les autres fonctionnalités du logiciel, en s'intégrant au sein de l'architecture de l'extension.

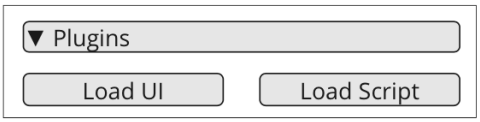

FIGURE 8 – Maquette de l'UI d'intégration de plugins.

Pour une vision plus globale et concise des US et de leurs critères d'acceptabilité, voir le tableau récapitulatif (Section 7, Figure 34).

4 Tests

Durant le processus de développement du projet, les tests ont été la base de l'implémentation de chacune des User Stories. Deux types de tests sont à mettre en évidence dans ce projet.

4.1 Monkey Testing

Le premier type de tests à prendre en compte concerne la robustesse de l'interface graphique (UI). L'idée ici est d'être certain que son utilisation permet de réaliser les appels aux algorithmes souhaités, avec les paramètres adéquats, tout en assurant le bon fonctionnement du logiciel et de l'extension. On souhaite donc vérifier qu'aucun bug n'apparaît lors de la manipulation de l'UI.

Pour ce faire, chaque tâche dédiée à l'UI a été testée avec la méthode du Monkey Testing. En d'autres termes, chaque élément de l'UI est utilisé dans différentes conditions d'utilisation, allant de réalistes à aberrantes pour couvrir tous les cas possibles, en cherchant la présence de bugs. Chacun des bugs mis en évidences grâce à cette méthode ont été traités de manière à assurer une utilisation stable en toutes conditions.

4.2 Unit Testing

La seconde gamme de tests à considérer s'applique aux appels d'algorithmes, écrits manuellement ou via les différentes bibliothèques utilisées dans le projet (*SimpleITK* et *Elastix*), avec les différents paramètres prévus. Ces fonctionnalités sont testées à l'aide de tests unitaires précisés par la suite.

Pour mettre en place ces tests, nous tirons avantage de la classe `ScriptedLoadableModuleTest` dont 3D Slicer dispose comme présenté en Section 5.1.1. En surchargeant la méthode `runTest`, nous pouvons configurer les appels aux méthodes individuelles qui testent chaque fonctionnalité lorsqu'on clique sur le bouton *Reload & Test*.

En outre, pour faciliter l'utilisation de cette classe de test, nous avons implémenté les méthodes utilitaires `resourcePath` et `assertVolumeEqual`. Elles permettent respectivement de récupérer le chemin absolu jusqu'à un fichier donné, et de vérifier que deux *VTK Volume* sont égaux.

4.2.1 Prétraitements

Sélection de ROI

`test_roi_selection`

Dans le cadre du test unitaire de la sélection de ROI, deux méthodes sont utilisées à la suite dans la classe `Widget`, `create_mask` et `select_roi`. Elles sont toutes les deux appeler dans ces tests, le résultat final étant testé en le comparant au résultat attendu généré préalablement.

Tout d'abord, une image 3D a été générée via *SimpleITK* et *NumPy*. Cette image correspond à une sphère dont les voxels intérieurs valent 2, sur un fond dont les voxels valent 0. Elle est aléatoirement bruité avec des valeurs de 1 afin de rendre le seuillage moins évident (Figure 9). Une version non bruitée est également générée avec des valeurs de 1 à l'intérieur de la sphère. Cette version correspond à la ROI de la sphère uniquement (Figure 10).

Ainsi, en faisant une sélection de ROI par seuillage avec une valeur de seuillage de 2, on devrait sélectionner uniquement la sphère (Figure 11), et non le fond ni le bruit (Figure 12).

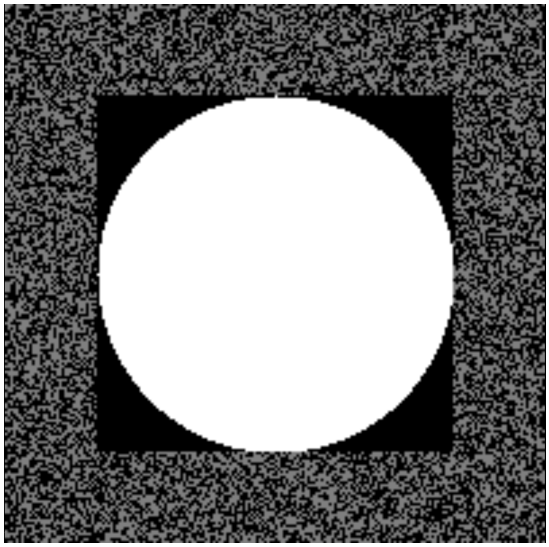

FIGURE 9 – Image générée de l'image bruitée de la sphère (image d'entrée).

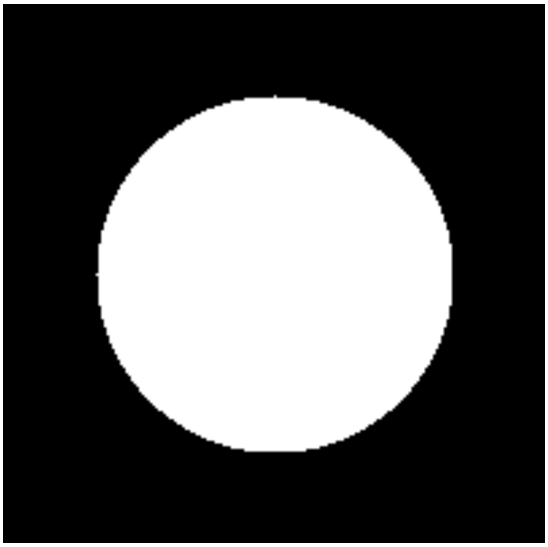

FIGURE 10 – Image générée de la ROI de la sphère, attendue après le calcul.

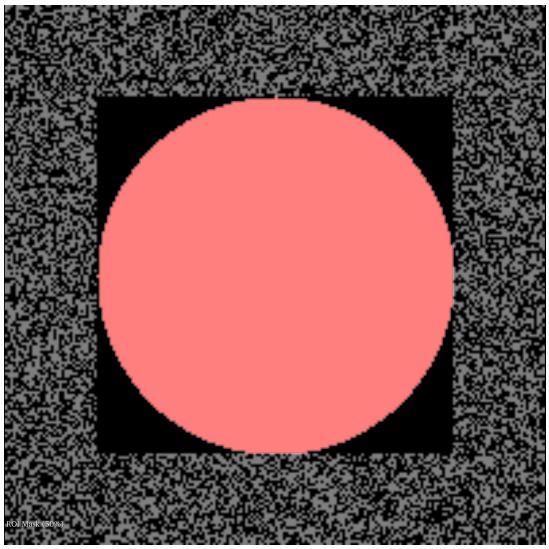

FIGURE 11 – prévisualisation de la ROI calculée avec un seuillage à 2.

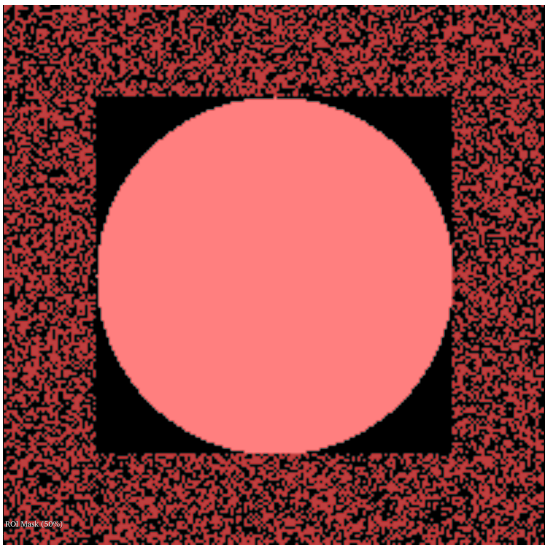

FIGURE 12 – prévisualisation de la ROI calculée avec un seuillage à 1.

Rognage

`test_manual_cropping`

Pour le test unitaire du rognage manuel qui teste la méthode `crop`, la première étape consiste à créer et stocker une image *VTK Volume* avec 3D Slicer. Cette image qui constituera l'image attendue, à partir d'une boîte englobante valide définie arbitrairement.

Puis dans la méthode, on commence par vérifier que des paramètres invalides résultent en une erreur en ajoutant ou soustrayant une grande à valeur à la boîte englobante définie au préalable.

Finalement, on crée une image à partir des paramètres valides et on s'assure qu'elle est identique à l'image attendue.

`test_automatic_cropping`

Le test unitaire pour le rognage automatique est très similaire à celui du rognage manuel. Il est construit de la même manière à la seule différence qu'il est également nécessaire de créer un *VTK Volume* représentant la ROI, et qu'il est question de tester les marges au lieu d'une boîte englobante.

Par ailleurs l'image de la ROI est créée à partir de notre extension. En effet, ce test admet que la sélection de ROI est fonctionnelle et, qu'on ne teste ici que le rognage automatique.

rééchantillonnage

`test_resampling`

La fonctionnalité de rééchantillonnage est testée en prenant plusieurs couples d'images avec des résolutions spatiales différentes, et en vérifiant que les deux aient la même en sortie.

Dans cette optique, plusieurs couples d'images d'entrée/cible sont définis grâce aux images de test. Pour chacun de ces couples, la méthode `Logic.crop` est appelée pour rééchantillonner l'image d'entrée en fonction de l'image cible. On vérifie alors que l'image en sortie de cette méthode est bien rééchantillonnée.

4.2.2 Recalage

Les tests peuvent être exécutés indépendamment de 3D Slicer, il suffit de lancer le script dédié `test_registration.py` disponible dans le dossier *Scripts/Registration*. Les tests s'effectuent sur un jeu de données disponibles dans le dossier *TestData* déjà utilisé pour les tests précédents. Il contient entre autre une image cible (référence fixe) *RegLib_ C01_ MRMeningioma_ 1*, une image à recaler *RegLib_ C01_ MRMeningioma_ 2*, une image rééchantillonnée, une image résultat d'un recalage affine, et des fichier .tfm symbolisant les résultats des transformations attendues, ce sont sur ces transformations que se font la plupart des tests.

Avant de pouvoir s'attaquer aux tests de chaque scripts. Il est nécessaire de tester les méthodes auxiliaires du fichier *Utilities.py* incessamment employés dans le code des méthodes de recalage.

`test_select_metrics_1`

La première méthode testée, intitulée *select_ metrics*, attache une métrique à l'objet *ImageRegistrationMethod*. Un recalage basique avec les méthodes de la bibliothèque SimpleITk résulte en un objet transform, directement comparé à un autre objet transform obtenue de manière similaire, mais la métrique n'a pas été sélectionné avec la fonction testée. Les transforms sont comparées à l'aide de leur coefficients. La métrique sélectionnée ici se nomme *Mattes Mutual Information*

`test_select_metrics_2`

Ce test reprend les grandes lignes du test ci-dessus, la seule différence se situe dans la métrique testée, ici *Mean Squares*. Les objets transforms sont comparés de la même manière.

`test_select_metrics_3`

Même chose qu'au dessus, mais cette fois nous testons la métrique *Joint Histogram Mutual Information*.

`test_select_metrics_4`

Même chose qu'au dessus, mais cette fois nous testons la métrique *Correlation*.

`test_select_metrics_5`

Ce test unitaire contient un ensemble de tests négatifs sur la méthode *select_ metrics*. Elle affirme aussi que différentes métriques ne donnent pas les mêmes résultats pour un jeu de paramètres équivalent. Pour cela on compare les transforms résultants des recalages.

`test_select_interpolator`

Ce test unitaire affirme la robustesse de la méthode *select_ interpolator*. Pour cela, la méthode testée fixe un interpolateur sur un objet *ImageRegistrationMethod*, puis on récupère l'interpolateur avec la méthode de la bibliothèque *GetInterpolator*, et il y a une simple assertion entre l'interpolateur sélectionné et l'interpolateur récupéré. Tous les interpolateurs disponibles dans l'extension sont testés de cette manière.

`test_select_optimizer_1`

Approche analogue à plus haut, la méthode *select_ interpolator* met au point l'interpolateur passé en paramètre à un objet de type *ImageRegistrationMethod*, les méthodes de la bibliothèque SITK prennent le relais pour élaborer un objet de type transform, un autre objet *ImageRegistrationMethod* est créé, cette fois sans utiliser la méthode testée, et s'exécute pour créer un objet transform, que l'on va comparer avec celui précédemment calculé. Cette méthode teste l'optimiseur *Gradient Descent*

`test_select_optimizer_2`

La même méthode de test unitaire du dessus est reprise, cette fois ci, l'optimiseur *Exhaustive* est mis à l'épreuve. Nous comparons les coefficients des deux objets *transforms* calculés, avec un delta de 0.01.

`test_select_optimizer_3`

LBFGSB mis au banc d'essais sur ce test unitaire. Nous comparons les coefficients des deux objets *transforms* calculés, avec un delta de 0.0001.

`test_select_optimizer_4`

Ici, des tests négatifs sont effectués avec les valeurs utilisateurs insensées. ce genre de valeurs ne peuvent arriver puisque l'UI prévient ce genre de comportement.

`test_rigid_registration_1`

Ce premier test exécute la méthode *rigid_ registration* du fichier *Rigid.py*, récupère la transformation obtenue et compare la transformation attendue du fichier *expected_ transform_ 1.tfm*. Ce sont les coefficients de la transformations qui sont comparés. Chaque tests rigides utilisent différents optimiseurs et différents paramètres pour prouver la robustesse du code.

`test_rigid_registration_2`

Ce test réalise la même chose, mais avec l'optimiseur exhaustif, et le fichier *expected_ transform_ 2.tfm*. A noter que pour le recalage rigide, la reproductibilité des résultats est possible, mais lors du passage au non rigide, il n'est pas possible d'avoir exactement les mêmes résultats. De ce fait nous ne comparerons plus les coefficients.

`test_rigid_registration_3`

Comme pour le premier et deuxième test, nous vérifions que la transformation résultante n'est pas vide. Cette fois-ci l'optimiseur LBFGSB est utilisé, et nous comparons au fichier contenant le résultat attendu *expected_ transform_ 3.tfm*.

`test_non_rigid_registration_1`

Les tests de recalage non rigide s'exécutent avec la méthode *non_ rigid_ registration* disponible dans le fichier *NonRigid.py*. Ce premier test utilise la méthode B-Spline avec l'optimiseur LBFGSB. La comparaison de *expected_ transform_ 4.tfm* avec les coefficients rassure sur le fait que la méthode fonctionne, même si les résultats ne sont pas tout à fait les mêmes à cause de la particularité du non rigide à déformer.

`test_non_rigid_registration_2`

La descente de gradient et le recalage non rigide n'assurent pas une reproductibilité, malgré tout, il y a comparaison du nombre de coefficients. On s'assure aussi que la transformation n'est pas vide.

`test_non_rigid_registration_3`

Ce test vérifie que l'utilisation d'un vecteur *Scale Factor* avec l'optimiseur LBFGSB lève bien une exception. Il n'y a pas plus de tests négatifs sur ces méthodes puisque les *observers* attachés à l'UI préviennent les entrées utilisateurs incohérentes.

`test_demons_registration_1`

C'est un simple test unitaire qui vérifie l'exécution de la méthode *non_ rigid_ registration* avec le filtre général Demons. Comme attendu, puisque les images n'ont pas été rééchantillonnées, l'objet transform est vide.

`test_demons_registration_2`

Le test unitaire 2 exécute la même méthode que le test précédent. L'exactitude de la méthode est testée via l'exécution dans le test d'une méthode Demons depuis la bibliothèque avec les mêmes paramètres. Les coefficients sont comparés avec un delta de 0.01, puisque la reproductibilité n'est pas assurée.

`test_demons_registration_3`

Ce test unitaire sélectionne Diffeomorphic Demons, et comme prévu donne un résultat vide même avec une image rééchantillonnée, en effet ce filtre a besoin d'un recalage affine avant d'être utilisée,

comme les autres filtres Demons. La méthode compare les coefficients avec la méthode de la bibliothèque directement, avec un delta de 1 pour les coefficients non fixes, et un delta de 0.1 pour les coefficients fixes.

`test_demons_registration_4`

Ce test est similaire au test précédent, seul le filtre Demons est altéré, c'est avec le filtre Fast Symmetric Forces Demons que la méthode de recalage s'exécute. Comme auparavant, l'objet *transform* est comparé à un autre objet *transform* directement obtenue depuis la bibliothèque, cette fois avec un delta de 0.01 pour tous les coefficients.

`test_demons_registration_5`

Ce dernier test sur la partie Demons reprend la même structure que son test antécédent. Cette fois Symmetric Forces Demons est testé, les coefficients sont comparés avec un delta de 0.01 chacun.

4.2.3 Carte de différence

`test_difference_map`

Le test se décompose en trois parties, une pour la différence de valeur absolue, une pour la différence de gradient, et enfin une dernière pour le noyau de convolution. Pour tester les fonctions de la carte de différence, nous avons créé deux images de référence de type **vtkMRMLScalarVolumeNode** à partir de tableau numpy de taille [3x3x3] appelé *imageData1* et *imageData1*, ces deux tableaux sont remplis de 0. Il suffit d'appeler la méthode *difference_ map*, et comparer l'image de sortie et l'image attendu.

Différence de la valeur absolue

Pour la différence de la valeur absolue, pour un voxel donné, sur les deux images d'entrées, comparer la valeur attendu. Dans notre cas nous avons juste mis sur *imageData1* un voxel à 0 et *imageData2* un voxel à 1. La valeur attendus de celui-ci est 1 car la valeur absolue de $|0 - 1|$ est 1.

Différence du gradient

Pour ce test, nous avons juste précalculé la valeur attendu du tableau grâce à la méthode de NumPy `np.gradient`. Ça nous permet aussi de savoir si la fonction arrive bien à passer de **vtkMRMLScalarVolumeNode** à `np.array` et inversement.

Noyau de convolution

Enfin, pour tester le noyau de convolution sur le voxel donné [1,1,1], nous avons crée une différence de 0.5, qui se traduit en une différence de 1 après normalisation. Le résultat attendu est le voxel [1,1,1] toujours sur une valeur de 1. Et ses voxels voisins (car noyau de 3) la moyenne des voxels contenu dans le noyau.

5 Implémentation

5.1 Implémentation globale

5.1.1 Module d'extension

3D Slicer dispose d'un outil permettant de générer un modèle d'extension vide depuis son interface même. Une extension peut être composée d'un ou plusieurs modules. Un module est une composante de l'extension qui sera affichée dans 3D Slicer indépendamment des autres — attention à ne pas confondre un module d'extension de 3D Slicer et un module Python qui correspond plus simplement à un script Python que l'on peut importer.

Dans notre cas, nous devons n'utiliser qu'un seul module d'extension afin de regrouper toutes les fonctionnalités au sein d'une interface minimale. Un seul script Python sera alors nécessaire pour intégrer à notre extension le module.

Le module d'extension est composé de quatre classes qui héritent chacune d'une classe mère appropriée provenant de 3D Slicer. En effet, `ScriptedLoadableModule` est le module Python de 3D Slicer contenant les quatre classes à implémenter via des classes filles. Dans notre cas, le module est nommé *Custom Registration*. Les quatre classes du module d'extension défini par le script `CustomRegistration.py` seront alors les suivantes.

`CustomRegistration(ScriptedLoadableModule)`

Classe principale définissant le module d'extension par un titre, des catégories, des dépendances potentielles, des collaborateurs, un texte d'aide informatif, un texte de reconnaissances et remerciements, et d'autres propriétés optionnelles. Son but est d'intégrer le module d'extension dans le logiciel et de le rendre exploitable en liant les trois autre classes à 3D Slicer.

`CustomRegistrationLogic(ScriptedLoadableModuleLogic)`

Classe abrégée `Logic` par la suite, dédiée aux algorithmes et calculs, ne manipulant pas l'UI ni la scène de 3D Slicer. Les méthodes sont faites pour être appelées par la classe suivante.

`CustomRegistrationWidget(ScriptedLoadableModuleWidget)`

Classe abrégée `Widget` par la suite, dédiée au lien entre l'UI et les algorithmes implémentés. Elle manipule la scène de 3D Slicer, interagit avec l'UI du panneau de traitements et exploite les algorithmes définis dans la classes `Logic`.

`CustomRegistrationTest(ScriptedLoadableModuleTest)`

Classe abrégée `Test` par la suite, dédiée à l'implémentation des tests unitaires du module d'extension. Elle peut être appelée directement depuis 3D Slicer afin de lancer les tests unitaires.

Nota Bene L'UI du panneau de configuration, manipulée par la classe `Widget`, est simplement définie par le fichier de ressource `Panel.ui`. Depuis les maquettes esquissées lors du développement du projet (Section 3.2), elle a été mise en jour en fonction des nouveaux besoins à chaque étape.

5.1.2 Architecture générale

Dans 3D Slicer, une image 3D est stockée dans un objet de type volume de la bibliothèque VTK. La scène correspond à l'environnement dans lequel les différents volumes évoluent. Autrement dit, lorsqu'un volume `volume` est créé, pour qu'il soit détecté par le logiciel, il doit être intégré à la scène via une méthode de 3D Slicer : `mrmlScene.AddNode(volume)`. Il en est de même pour la suppression. Supprimer les références au volume ne suffit pas, il faut avant tout le retirer de la scène grâce à une méthode dédiée : `mrmlScene.RemoveNode(volume)`. Ces notions de volumes et de scène sont importantes pour la compréhension de l'architecture générale.

Dans la classe `Widget`, le module d'extension possède trois méthodes principales qui permettent de gérer le module d'extension dans son entièreté et d'assurer son fonctionnement.

`setup`

Méthode qui configure le module d'extension. Elle est appelée automatiquement au lancement de ce dernier. Concrètement, elle charge l'UI du panneau de traitements, configure chaque fonctionnalité, puis initialise des écouteurs d'évènements sur la scène de 3D Slicer afin de mettre à jour le module d'extension si celle-ci est modifiée dans 3D Slicer (cela permet d'intégrer notre module d'extension au sein du logiciel).

`reset`

Méthode qui rétablit l'état par défaut du module d'extension. Elle est appelée automatiquement lorsque le module est rechargé (elle peut aussi être appelée manuellement). Elle se charge de rétablir l'état par défaut de chaque fonctionnalité.

`update`

Méthode qui met à jour le module d'extension en fonction de la scène actuelle. Elle est appelée par `setup` pour initialiser le module avec la scène initiale et par les écouteurs d'évènements mis en place dans cette même méthode pour actualiser le module d'extension en temps réel (elle peut aussi être appelée manuellement). Tout d'abord, elle récupère la liste des volumes correspondants à des images dans la scène. Elle met à jour individuellement chacune des fonctionnalités en suivant. Précision importante, cette méthode peut être bloquée manuellement. En effet, les écouteurs d'évènements peuvent détecter des mises à jour de la scène de 3D Slicer provenant du module d'extension et qui ne nécessitent pas de mise à jour de l'extension. Il suffit alors de bloquer la mise à jour à l'aide de la variable booléenne dédiée `update_allowed` avant la modification de la scène, puis de la débloquer après. Cela permet d'éviter les mises à jour inutiles, et certaines boucles infinies dans certains cas (en particulier lors de prévisualisations nécessitant des images temporaires dont nous parlerons plus tard).

5.1.3 Architecture des fonctionnalités

Chaque fonctionnalité est implémentée en appliquant le même principe afin de s'intégrer complètement à notre module d'extension (les trois mêmes méthodes sont implémentées). On désignera ici une fonctionnalité quelconque avec `<feature>`, l'architecture suivante étant valable pour toute fonctionnalité du module d'extension.

`setup_<feature>`

Méthode qui configure la fonctionnalité. Elle est appelée au lancement du module d'extension, par la méthode `setup` générale. Elle récupère les éléments de l'UI que l'utilisateur peut manipuler afin de

pouvoir en extraire les paramètres des algorithmes ou les manipuler directement (valeurs par défaut, minimum, maximum). Elle initialise également les données qu'elle va manipuler en mémoire, en créant les variables nécessaires et en initialisant les valeurs par défaut via la méthode suivante.

`reset_<feature>`

Méthode qui rétablit l'état par défaut de la fonctionnalité. Elle est appelée tout d'abord par la méthode `setup_<feature>`, par la méthode `reset` générale et dès que nécessaire à travers le module d'extension. En particulier, cette méthode réinitialise les paramètres des algorithmes et l'UI dans le panneau de traitements, puis elle met à jour la fonctionnalité en appelant la méthode suivante.

`update_<feature>`

Méthode qui met à jour la fonctionnalité en fonction de la scène actuelle ainsi que des données du module d'extension. Elle est appelée par la méthode `reset_<feature>`, par la méthode `update` générale et dès que nécessaire à travers le module d'extension.é

5.1.4 Images d'entrée/cible et visualisation

Dans le but d'intégrer au maximum l'extension au sein de 3D Slicer et de ne pas modifier son comportement général, nous avons fait le choix d'utiliser les vues déjà disponibles dans la disposition par défaut. La seule exception est la vue 3D qui est masquée par défaut mais peut être réaffichée. Il est donc possible de manipuler le logiciel en toute fluidité à travers les différents modules et les différentes vues, tout en utilisant notre extension.

Notre extension étant constituée de fonctionnalités de traitements d'une image indépendante (sélection de ROI, rognage manuel et automatique) ou d'une image en fonction d'une autre (rééchantillonnage automatique, recalage rigide et non-rigide, calcul de carte de différence), son architecture repose sur un système de choix d'image d'entrée (c'est-à-dire l'image à traiter) et d'image cible (c'est-à-dire l'image de référence pour appliquer les traitements relatifs sur l'image d'entrée).

Les vues disponibles dans 3D Slicer sont alors déliées pour ne plus afficher une seule image sous différents axes, mais plutôt les deux images sélectionnées, ainsi qu'une carte de différence une fois calculée. Il est donc possible de visualiser l'image d'entrée comme image principale dans la vue rouge (en haut à gauche) et l'image cible dans la vue verte (en bas à gauche), la vue jaune (en bas à droite) étant dédiée à la carte de différence.

La vue 3D (en haut à droite) masquée par défaut peut-être réaffichée en activant le mode *Pascal Only*, et l'image d'entrée sélectionnée y sera affichée en 3D (elle restera vide si aucune image n'est sélectionnée comme image d'entrée).

Lors de l'application d'un traitement sur l'image d'entrée, celle-ci est conservée dans la scène. Une nouvelle image correspondant à l'image traitée est créée et ajoutée à la scène, et est automatiquement sélectionnée comme nouvelle image d'entrée afin d'afficher le résultat directement dans la vue rouge dédié (en haut à gauche).

La vue générale du module d'extension est visible sur la Figure 13. La Figure 14 met quant à elle en évidence l'UI du système de choix d'images d'entrée et cible. La vue générale et l'UI du système de choix d'images d'entrée et cible sont basées sur la maquette établie lors de la conception (Figure 2).

Pour mettre en place le système d'images d'entrée/cible, deux séries de méthodes ont été implémentées : l'une pour la gestion de l'image d'entrée, l'autre pour celle de l'image cible. Les deux séries sont dédiées aux mêmes tâches, mais appliquées aux deux cas bien distincts avec leurs propres particularités.

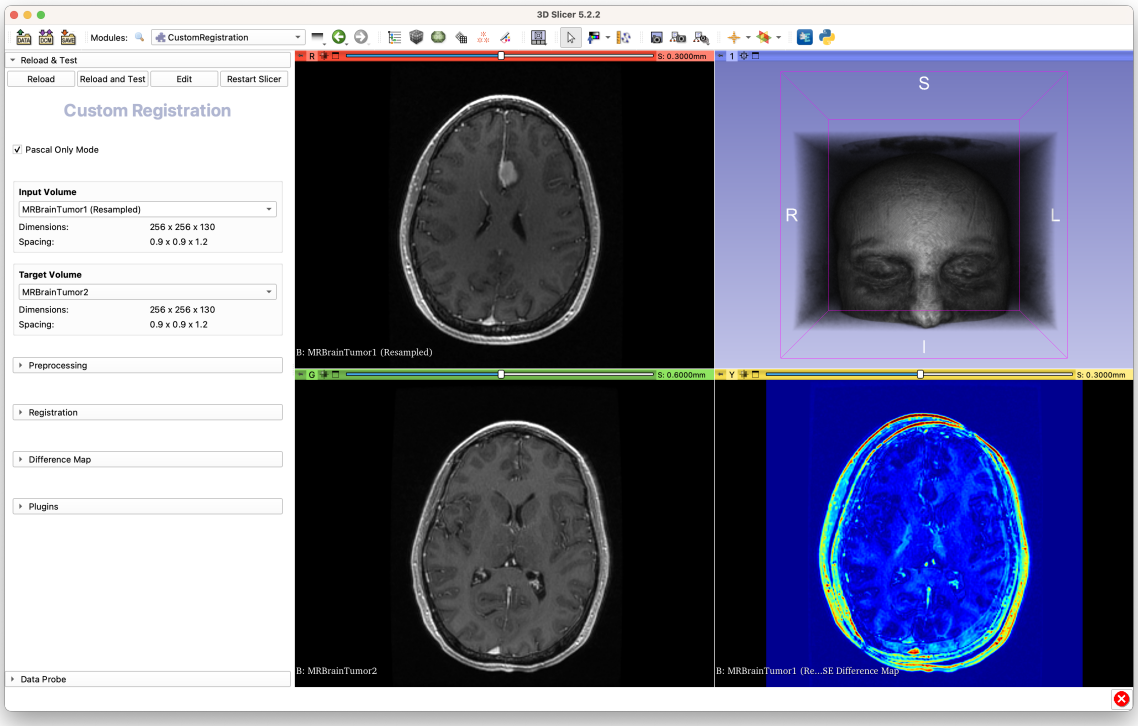

FIGURE 13 – Aperçu général du module d'extension basé sur la maquette (Figure 2), comprenant le panneau de traitements à gauche et les différentes vues à droite (en haut à gauche la vue 2D de l'image d'entrée, en haut à droite la vue 3D optionnelle du mode Pascal Only, en bas à gauche la vue 2D de l'image cible, en bas à droite la vue 2D de la carte de différence.

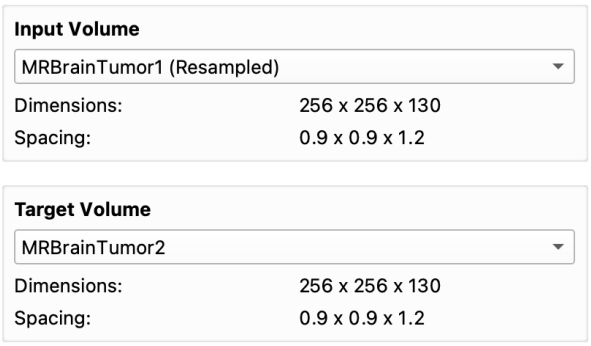

FIGURE 14 – Aperçu de l'UI du système de choix d'image d'entrée (Input Volume) et d'image cible (Target Volume) basé sur la maquette (Figure 2), comprenant les dimensions et la taille des voxels dans les trois dimensions des images.

`setup_<intput/target>_volume`

Méthode qui configure le système dédié à l'image d'entrée/cible, au même titre que `setup_<feature>` vu précédemment. Elle est donc appelée par la méthode `setup` générale et se charge d'initialiser les données et de récupérer les éléments de l'UI. Concrètement, elle récupère les listes déroulantes de sélection d'image et y insère les actions sur les volumes disponibles (renommage et suppression). Elle lie ensuite les actions à leurs évènements respectifs (méthodes associées à la sélection, au renommage et à la suppression détaillées ci-dessous).

`reset_<input/target>_volume`

Méthode qui rétablit l'état par défaut du système dédié à l'image d'entrée/cible, au même titre que `reset_<feature>` vu précédemment. Ici, il suffit alors de désélectionner l'éventuelle image d'entrée/cible grâce à la méthode suivante.

`select_<input/target>_volume`

Méthode qui sélectionne une image comme image d'entrée/cible par son identifiant dans la liste des images. Si l'identifiant est -1 (valeur par défaut), aucune image ne sera sélectionnée (il s'agit de l'état par défaut). Elle est appelée lorsque la liste déroulante change de valeur (c'est-à-dire que l'utilisateur change d'image sélectionnée), ainsi que par `reset_<input/target>_volume` (avec l'identifiant -1). La référence en mémoire de l'image d'entrée/cible est alors adaptée, et la méthode `update` générale est appelée afin de mettre à jour le module d'extension en fonction.

`rename_<input/target>_volume`

Méthode qui gère l'évènement du renommage de l'image d'entrée/cible actuellement sélectionnée. Cet évènement est déclenché lorsque l'utilisateur choisi l'action de renommage depuis la liste déroulante (si une image est sélectionnée comme image d'entrée/cible). Le nom de l'image d'entrée/cible est alors modifié et le module d'extension actualisé.

`delete_<input/target>_volume`

Méthode qui gère l'évènement du suppression de l'image d'entrée/cible actuellement sélectionnée. Cet évènement est déclenché lorsque l'utilisateur choisi l'action de suppression depuis la liste déroulante (si une image est sélectionnée comme image d'entrée/cible). L'image d'entrée/cible est alors supprimée de la scène, et ses références sont remises à `None`. Aucune image n'est alors sélectionnée.

Le système de vues a été mis en place en suivant la même architecture que les fonctionnalités (Section 5.1.3), à l'exception d'une méthode.

`setup_view`

Méthode qui configure le système de vues. Elle est appelée par la méthode `setup` générale. Elle récupère les vues afin de pouvoir les manipuler par la suite, puis les initialise grâce à la méthode suivante.

`reset_view`

Méthode qui rétablit l'état par défaut des vues. Elle est appelée par `setup_view`, par la méthode `reset` générale et dès que nécessaire à travers le module d'extension. Elle désactive l'affichage de tous les volumes avant tout, puis met à jour la vue selon les données actuelles via la méthode suivante.

`update_view`

Méthode qui met à jour les vues en fonction de la scène actuelle, et des images sélectionnées comme images d'entrée et cible. Elle est appelée par la méthode `reset_view`, par la méthode `update` générale et dès que nécessaire à travers le module d'extension. Elle utilise la méthode suivante pour afficher les images sélectionnées dans les vues appropriées.

`update_specific_view`

Méthode qui, comme son nom l'indique, met à jour une vue spécifique dont l'identifiant est spécifié en paramètre, avec la référence d'une image, et éventuellement la référence d'un volume à afficher en transparence par dessus l'image choisie (`mask`).

5.2 Prétraitements

5.2.1 Sélection de ROI

La sélection de ROI est un outil n'affectant pas l'image directement. Il s'agit seulement de sélectionner une zone spécifique dans l'image contenant des informations pertinentes pour l'analyse ou le traitement ultérieur.

La sélection de ROI peut être utilisée pour faciliter les traitements des images, en particulier le recalage, en identifiant les zones d'intérêt communes à plusieurs images. Une fois la ROI sélectionné, elle peut être utilisée comme référence pour le traitement/l'alignement des images, permettant d'assurer que les zones d'intérêt correspondent bien les unes aux autres dans toutes les images.

Suivant la US I, la sélection de ROI est implémentée grâce à la méthode du seuillage binaire. L'implémentation s'est donc faite en deux étapes.

Une première méthode de la classe `Logic`, `create_mask`, se charge d'effectuer une binarisation de l'image passée en paramètre au format *VTK Volume*, grâce à une valeur de seuillage également passée en paramètre. L'image est tout d'abord convertie au format *SimpleITK Image* afin que le seuillage puisse être effectué grâce à la méthode de *SimpleITK* `BinaryThreshold`. La méthode `create_mask` retourne alors la nouvelle image correspondant à cette binarisation, convertie au format *VTK Volume*.

Toujours dans la classe `Logic`, une deuxième méthode, `select_roi`, prend en paramètre l'image sur laquelle la ROI doit être sélectionnée et la masque calculée par la première méthode, les deux au format *VTK Volume*. Ils sont avant tout convertis au format *SimpleITK Image*. La méthode se charge alors de récupérer la plus grande composante connexe du masque (l'image binarisée). Cette nouvelle version de l'image correspond alors à la ROI. Celle-ci est convertie au format *VTK Volume* à nouveau afin de pouvoir être manipulée dans la scène de 3D Slicer.

Il est possible de sélectionner une ROI sur une image simplement, ou bien en parallèle sur deux images, de la même manière sur l'interface dédiée. L'utilisateur doit avoir préalablement sélectionné une image comme image d'entrée et/ou une comme image cible via l'interface supérieure.

Suivant la maquette établie lors de la conception (Figure 3) et appliquant cette possibilité de sélection les ROI de deux images distinctes, dans la section *ROI Selection*, deux curseurs coulissants (*sliders*) sont présents pour permettre à l'utilisateur de choisir une valeur de seuillage sur chacune des images. Ces *sliders* sont liés à la méthode `Logic.create_mask` pour sauvegarder en mémoire les masques. Le bouton *Select ROI* est lié à la méthode `Logic.select_roi`. Au clic sur le bouton *Select ROI*. La ROI ainsi calculée est sauvegardée dans un dictionnaire liant les images et leur ROI, `volume_roi_map`. L'interface est représentée en Figure 15.

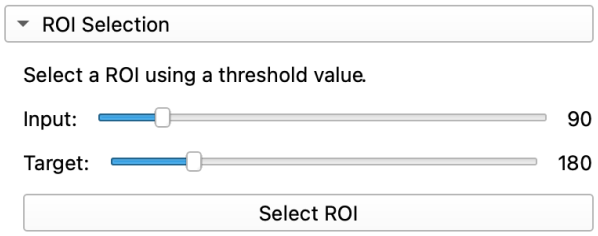

FIGURE 15 – Aperçu de l'interface de sélection de ROI.

Prévisualisation et visualisation finale

Afin d'améliorer l'ergonomie de la fonctionnalité de sélection de ROI, nous avons intégré des options de prévisualisation et de visualisation. Elles permettent à l'utilisateur de visualiser la région de l'image désignée respectivement avant et après validation de la sélection. Ainsi, l'utilisateur peut ajuster le seuil pour choisir la région adéquate pour son application.

La prévisualisation est générée dans la méthode `preview_roi_selection` de la classe `Widget`. Cette méthode a l'avantage de gérer plusieurs variations. En effet, comme la sélection de la ROI peut être effectuée sur l'image d'entrée et/ou l'image cible, notre méthode offre trois options : `"input"`, `"target"` ou `"all"` qui permettent de générer une prévisualisation adéquatement au cas voulu.

L'approche adoptée pour illustrer la ROI consiste afficher sur la vue concernée le *VTK Volume* en avant-plan transparent et d'attribuer la couleur rouge aux voxels lors de l'affichage, grâce à une *Color Table* (classe `vtkMRMLColorTableNode`).

La première étape du processus général est de supprimer le précédent affichage s'il existe. Ensuite, il convient de récupérer le seuil entré par l'utilisateur grâce au `slider` adéquat. À partir de celui-ci et de l'image sélectionnée comme entrée/cible, on peut récupérer le masque représentant la binarisation grâce à la méthode `Logic.create_mask` mentionnée précédemment. Il convient de noter que l'intervalle des sliders qui permettent de choisir un seuil est configuré pour correspondre à la plage dynamique de l'image traitée afin d'éviter les erreurs de valeurs. Par la suite, le *Display Node* (classe `vtkMRMLScalarVolumeDisplayNode`) — permettant de paramétrer l'affichage de l'image — est récupéré afin de lui assigner la *Color Table* définie au préalable. Il suffit ensuite d'ajouter le *VTK Volume* représentant la ROI à la scène. Enfin, il convient d'afficher le masque par-dessus l'image traitée (*foreground*), en transparence (50%), afin de pouvoir visualiser convenablement la zone sélectionnée (voir Figure 16).

Quant à la visualisation, elle se passe dans la méthode `display_roi` de la classe `Widget`. Elle fonctionne exactement de la même manière que `preview_roi_selection`. La différence réside seulement dans la couleur d'affichage qui est cette fois verte (voir Figure 17), et le *VTK Volume* utilisé qui n'est pas la binarisation sauvegardée en mémoire, mais la ROI stockée dans le dictionnaire `volume_roi_map`.

La mise à jour de la prévisualisation et de la visualisation finale se fait dans une seule et même méthode `Widget.update_roi_selection`. Elle est sollicitée chaque fois que l'utilisateur bouge un *slider*.

De surcroît, puisqu'une vue sur 3D Slicer ne peut afficher que deux images à la fois, elle s'occupe d'afficher la prévisualisation si le panneau de la sélection de ROI est ouvert, la visualisation finale sinon (si elle existe).

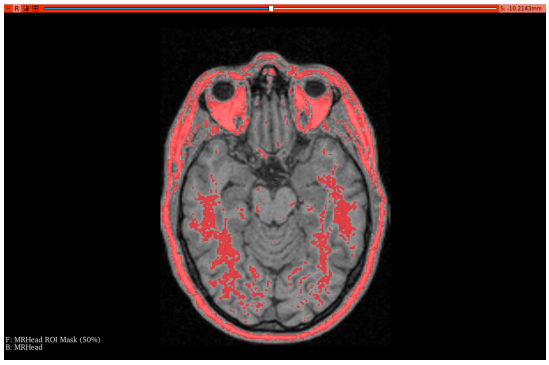

FIGURE 16 – Exemple de prévisualisation de sélection de ROI (avec un seuil `Threshold=90`) sur une IRM d'une tête (échantillon disponible sur 3D Slicer), depuis un angle axial.

FIGURE 17 – Exemple de visualisation de sélection de ROI suite à la sélection de la Figure 16.

Limitations

La sélection de ROI telle qu'implémentée présente cependant plusieurs limitations.

Tout d'abord un bug est connu mais sa réparation n'a pas pu se faire par manque de temps. En effet, si l'on tente de faire une double sélection de ROI (images d'entrée/cible en parallèle) mais sur la même image, le processus ne s'arrête pas car il n'y a pas d'erreurs, mais les références en mémoire sont erronées pour l'une des sélections (il n'y a qu'une seule ROI associée à l'image, alors qu'il y a deux visualisations avec des références distinctes). Le résultat est une visualisation correcte, mais une autre affichant le mauvais volume en avant-plan transparent, comme on peut le voir en Figure 18.

FIGURE 18 – Bug de la double sélection de ROI en parallèle sur la même image.

En outre, cette méthode de sélection de ROI est limitée dans son aptitude à sélectionner précisément une ROI. En effet, l'utilisateur n'a pas de contrôle sur la ROI sélectionnée, seulement sur le seuillage préalable. Or, il pourrait voudrait choisir un tissu unique en particulier par exemple. Cette méthode permet cependant une simplicité et rapidité d'utilisation.

Enfin, au niveau du code, il serait pertinent de simplifier les appels aux méthodes en factorisant `Widget.preview_roi` et `Widget.display_roi` qui se chargent toutes les deux d'afficher des volumes en avant-plan transparent, mais dans deux cas différents.

5.2.2 Rognage

Le rognage est une opération de traitement d'image couramment utilisée pour découper une image en supprimant les parties inutiles, telles que les bords noirs, les zones de fond ou les parties qui ne sont pas pertinentes pour l'analyse.

Cette opération est notamment utile en amont du recalage, afin de réduire la taille de l'image et de faciliter le traitement, tout en conservant les zones d'intérêt pour l'analyse. En éliminant les parties inutiles de l'image, le rognage peut améliorer la précision du recalage en réduisant les artefacts dus aux zones inutiles ou incohérentes de l'image. Le rognage peut également aider à minimiser les erreurs de recalage en réduisant les variations dans les régions environnantes des zones d'intérêt. En outre, la réduction de la taille de l'image peut également améliorer les performances du recalage en réduisant le temps de calcul et la consommation de mémoire.

Rognage manuel

L'implémentation de l'algorithme de rognage manuel a été réalisée dans la méthode `crop` de la classe `Logic`. Cette méthode requiert trois paramètres en entrée : une image 3D au format *VTK Volume* ainsi que deux tableaux de taille 3, décrivant les positions de début et de fin de l'image résultante dans le repère de l'image d'entrée.

Pour commencer, l'image d'entrée est convertie en image *SimpleITK*. Puis pour procéder au rognage, la classe `CropImageFilter` de la bibliothèque *SimpleITK* est utilisée. Cette classe permet de créer un filtre en spécifiant les marges à enlever depuis le début et la fin des bords de l'image à rogner. Ensuite, la méthode `Execute` permet d'extraire une région de l'image 3D en fonction de ces paramètres. Puis on attribue à l'image rognée l'origine, l'espacement et la matrice de direction de l'image d'entrée afin que les deux images conservent les mêmes propriétés. Finalement, l'image 3D rognée est convertie au format *VTK Volume* avant d'être retournée en tant que résultat final.

La gestion de ce traitement se fait dans la méthode `manual_crop` de la classe `Widget`, qui est appelée lorsque l'utilisateur clique sur le bouton *Crop*. Son rôle est principalement de sauvegarder l'image rognée en l'ajoutant à la liste des images 3D de la scène, puisqu'on verra par la suite que l'image rognée existe déjà avant que l'utilisateur ne clique sur le bouton *Crop* (voir paragraphe prévisualisation).

En outre, elle se charge de gérer les erreurs de paramètres. Si l'utilisateur entre des paramètres invalides (une position de début supérieure ou égale à la position de fin) et confirme le traitement avec le bouton *Crop*, une fenêtre avec un message d'erreur apparaîtra.

Rognage automatique

Le principe du rognage automatique est d'utiliser une ROI sélectionnée en amont pour définir une première zone à rogner, et d'ajouter une marge autour de celle-ci. Cette marge supplémentaire permet de

tenir compte de certaines imperfections dans la sélection de la ROI, tout en garantissant que l'ensemble de l'objet d'intérêt soit bien présent dans l'image finale.

De la même manière que pour le rognage manuel, l'implémentation de l'algorithme de rognage automatique se fait dans la méthode `automatic_crop` de la classe `Logic`.

À la différence de la méthode `manual_crop`, elle nécessite en entrée deux images 3D au format *VTK Volume* : l'image à rogner et l'image représentant la ROI grâce à des valeurs binaires. Elle demande également un tableau de taille 3 décrivant la taille des marges à ajouter autour de la boîte englobante de la ROI. Par la suite, elle récupère la boîte englobante de la ROI, (à ne pas confondre avec la boîte englobante de l'image représentant la ROI) avant d'y ajouter les marges. Pour finir, la méthode `crop` de la classe `Logic` génère l'image 3D rognée finale. Cette image est retournée avec sa boîte englobante (dans le repère de l'image d'entrée) à l'aide d'un tuple.

Toujours en suivant le même schéma que pour le rognage manuel, nous avons implémenté une méthode `automatic_crop` dans la classe `Widget` qui est exécutée dès lors que l'utilisateur clique sur le bouton *Automatic Crop*.

Elle s'occupe donc, de la même façon, de charger dans la scène l'image rognée, générée en amont dans la méthode `preview_automatic_crop` (voir paragraphe prévisualisation).

Elle gère également les erreurs de paramètres (marges dépassant les limites de l'image d'entrée) entrés par l'utilisateur, en affichant une fenêtre avec un message d'erreur.

Notons que pour utiliser ce mode de rognage, l'utilisateur doit avoir sélectionné une ROI au préalable. Autrement, une fenêtre de message d'erreur s'affichera pour lui signaler de le faire.

Paramétrage

En premier lieu, afin de choisir un mode de rognage, nous proposons des boutons d'option *Manual* et *Automatic* (voir Figures 19 et 20). Pour plus de lisibilité, seule l'UI du mode sélectionné est affichés.

Pour rogner une image manuellement, l'utilisateur a à sa disposition des zones de saisie numérique à incrémentation (voir Figure 19) qui lui permettent d'entrer les bords de l'image rognée souhaitée, (positions de début et de fin, qu'on appellera par la suite boîte englobante) dans le repère de l'image d'entrée. Par ailleurs, ces zones de saisie sont paramétrées de telle sorte qu'on ne puisse pas entrer des positions en dehors des dimensions de l'image d'entrée.

Pour rogner une image automatiquement, l'utilisateur dispose également de zones de saisie numérique à incrémentation (Figure 20). Celles-ci permettent d'entrer la taille des marges à ajouter autour de la sélection de ROI. Contrairement à celles du rognage manuel, les valeurs maximales des zones de saisie ne sont pas définies en fonction des limites maximales autorisées (le minimum est paramétré à 0 par défaut, ce qui nous convient). Cette situation est prise en compte ultérieurement lors de l'opération de sauvegarde (voir paragraphe Rognage automatique).

Prévisualisation

Dans l'optique de faciliter l'utilisation du rognage, nous avons ajouté une fonctionnalité de prévisualisation. Cette fonctionnalité permet à l'utilisateur de visualiser la région de l'image qui sera rognée avant d'exécuter le rognage. Ainsi, l'utilisateur peut mieux sélectionner la région souhaitée pour son application, en évitant les erreurs potentielles de rognage.

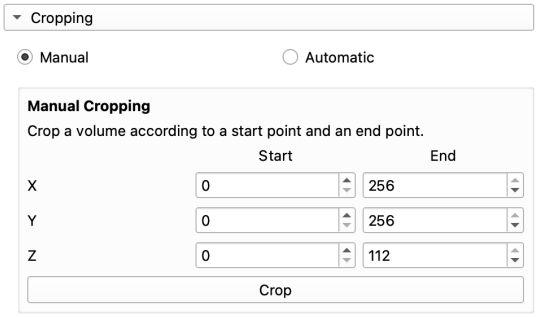

FIGURE 19 – Aperçu de l'UI de rognage manuel.

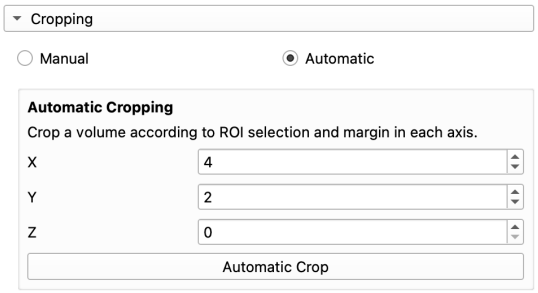

FIGURE 20 – Aperçu de l'UI de rognage automatique.

Ce processus a lieu, de manière similaire, dans les deux méthodes dédiées de la classe `Widget`, `preview_manual_cropping` et `preview_automatic_cropping`.

L'idée est de créer un objet *ROI Markup Node* (de type `vtkMRMLMarkupsROINode`) qui représente en fait une boîte semi-transparente, de couleur rouge par défaut (voir Figure 21). Le fait que ce soit une boîte et non un rectangle (autrement dit un objet 3D et non un objet 2D) permet à l'utilisateur de prévisualiser la zone à rogner sur toutes les *slices* et sous les trois vues proposées par 3D Slicer (axiale, sagittale et coronale).

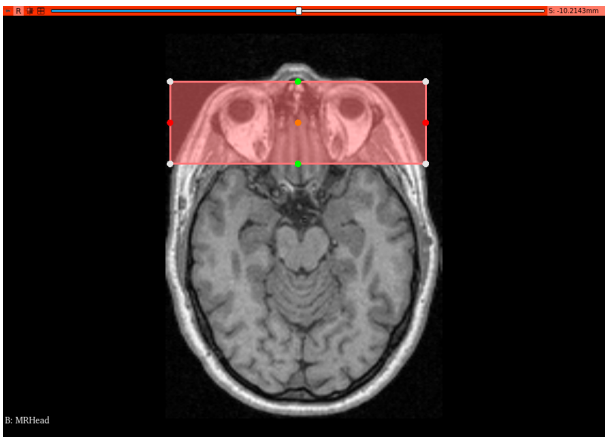

FIGURE 21 – Exemple de prévisualisation du rognage manuel (avec les paramètres `Start=(40, 0, 10)` et `End=(90, 200, 130)`) sur une IRM d'une tête (échantillon disponible sur 3D Slicer), depuis un angle axial.

Le procédé consiste dans un premier temps de supprimer le *ROI Markup Node* précédent s'il existe et de calculer le nouveau en appelant `preview_manual_cropping` (ou `preview_automatic_cropping`). Puis, à effectuer le rognage sur l'image choisie par l'utilisateur (*Input Volume*, Figure 14) avec la méthode `Logic.crop` (ou `Logic.automatic_crop`), à partir des paramètres entrés par l'utilisateur. Si les paramètres sont invalides, il n'y aura aucun message d'erreur mais la prévisualisation ne s'affichera pas. S'il n'y a pas d'erreur, la boîte englobante de l'image 3D rognée dans le repère global est ensuite récupérée. De la même manière que lorsqu'on utilise la méthode utilitaire `transfer_volume_metadata` après avoir créé le *VTK Volume* pour l'image rognée, on affecte au *ROI Markup Node* les mêmes propriétés que l'image originale. Cela permet de bien dimensionner la boîte de prévisualisation, et de la transformer (du repère local de l'image d'entrée) vers le repère global. Contrairement à un *VTK Volume*, on ne peut pas assigner directement les méta-données à un *ROI Markup Node*. C'est pourquoi nous sommes contraints d'appliquer les transformations directement sur la boîte englobante récupérée précédemment.

La gestion de la prévisualisation se fait dans la méthode `update_cropping` (de la classe `Widget`), indépendemment du mode de rognage choisi. Elle est appelée dès lors que l'utilisateur modifie un paramètre.

Dans la même idée que pour la sélection de ROI (voir Section 5.2.1), elle gère l'affichage du *ROI Markup Node*. Lorsque l'utilisateur ferme le panneau dépliant du rognage, la prévisualisation associée reste en mémoire et demeure cachée. Toutefois, cette dernière réapparaît lorsque le panneau est ouvert à nouveau.

Limitations

Finalement, notre implémentation comporte des limites au niveau de la prévisualisation. En effet, le calcul du *ROI Markup Node* n'est pas optimisé. Même si l'utilisateur ne valide pas le rognage, il est quand même appliqué sur l'image 3D afin de générer la prévisualisation, et ce à chaque fois qu'un paramètre est modifié. Notons tout de même que l'affichage reste en temps réel et ne nuit pas à l'utilisation de la fonctionnalité.

De plus, les points de contrôle du *ROI Markup Node* ne sont pas interactifs, ce qui aurait pu être pratique pour que l'utilisateur puisse ajuster manuellement la zone à rogner.

5.2.3 Rééchantillonnage

Le rééchantillonnage est une opération de traitement d'image permettant de modifier la résolution spatiale d'une image en modifiant la taille des voxels ainsi que les dimensions de l'image.

En général, le rééchantillonnage est nécessaire lorsque l'on souhaite aligner plusieurs images 3D acquises avec différents protocoles d'imagerie, ou lorsque l'on souhaite comparer des images avec différentes résolutions spatiales. Il s'agit donc d'une opération nécessaire avant un recalage ou un calcul de carte de différence.

Ce rééchantillonnage peut se faire par interpolation, c'est-à-dire en utilisant des méthodes mathématiques pour estimer les valeurs des nouveaux voxels manquantes entre les voxels existants.

Dans notre optique de traitement minimal suivant la US I, le rééchantillonnage est conçu de sorte de rééchantillonner l'image d'entrée en fonction de l'image cible.

L'interface dédiée est très simple et suit complètement la maquette prévue lors de la conception (Figure 5). Cette interface est représentée par la Figure 22.

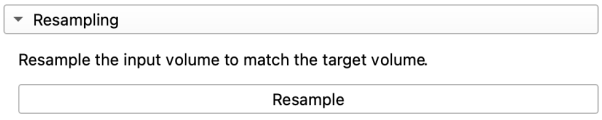

FIGURE 22 – Aperçu de l'UI de rééchantillonnage.

L'implémentation de l'algorithme de rééchantillonage se fait dans la méthode `Logic.resample`. Cette méthode prend en paramètres les images d'entrée et cible au format *VTK Volume*.

Le volume rééchantillonné est créé grâce à la méthode `Resample` de la bibliothèque `SimpleITK`, en utilisant l'interpolateur par défaut. Les deux volumes *VTK* sont convertis en image *SimpleITK* au préalable, et l'image *SimpleITK* sortie de est convertie au format *VTK*, et les méta-données de ce type d'objets sont mises à jours pour correspondre à celles de l'objet *SimpleITK*. Le volume rééchantillonné est alors retourné, l'algorithme étant terminé.

Dans la méthode `Widget.resample`, appelée lors du clic sur le bouton *Resample*, se charge d'appeler `Logic.resample` avec les images d'entrée et cible sélectionnées par l'utilisateur, et d'ajouter l'image résultante à la scène de 3D Slicer.

5.3 Recalage

D'après Betrouni (2009), "le recalage en imagerie médicale est une technique qui permet de superposer des images provenant de sources différentes ou de moments différents, afin de faciliter leur analyse ou leur interprétation". C'est un outil essentiel pour surveiller l'évolution des patients, et rendre un diagnostique plus complet. Nous nous concentrons sur trois types de recalage.

- **Rigide** Suppose que les images sont liées par une rotation et une translation. Il préserve la forme et la taille des objets dans les images. Il contient 6 degrés de liberté.
- **Affine** Permet la mise à l'échelle et le cisaillement des images en plus de la rotation et de la translation. Il préserve les lignes parallèles, mais pas les angles ni les distances. Il contient 10 degrés de liberté.
- **Non-rigide** Permet de gérer les déformations locales des images qui ne sont pas prises en compte par les transformations rigides ou affines.

L'objectif est d'intégrer au sein d'une même interface divers algorithmes de recalage disponibles sur deux bibliothèques principalement : SimpleITk et Elastix. L'attente principale avec cette demande est l'accès aux outils de recalage, la comparaison de ces outils et de leur résultat. Cette comparaison pourra être effectué in fine avec des algorithmes de recalages personnalisés directement implémentables par l'utilisateur.

5.3.1 Implémentation logicielle

L'implémentation de l'algorithme de recalage a été réalisée dans la méthode `register` de la classe `Logic`. Cette méthode se sert des images d'entrée et cible sélectionnées par l'utilisateur, c'est-à-dire l'image à recaler et l'image de référence (fixe).

Une vérification des paramètres utilisateur est d'abord effectuée à l'entrée de la méthode. Selon le type de recalage adopté (SimpleITK, Elastix), la suite s'effectue via `custom_script_registration` ou `elastix_registration`.

Méthode custom_script_registration

Il s'agit d'une méthode auxiliaire utile à l'exécution d'un script python externe en tâche de fond via l'extension Slicer Parallel Processing. Deux scripts Python sont livrés avec notre extension : `Rigid.py` et `NonRigid.py`. Tous deux intègrent des méthodes de recalage avec en entrées une image à recaler, une image fixe, et un dictionnaire propre à notre implémentation. Ce dictionnaire contient les paramètres pour chaque élément lié au recalage.

`Rigid.py` agrège les types de recalages rigide et affine. Des méthodes utilitaires implémentées dans le fichier `Utilities.py` aident à la création de l'objet SimpleITK `ImageRegistrationMethod`, à l'aide du dictionnaire d'entrée, c'est cet objet qui se charge du recalage, via la méthode `Execute`. Les recalages de chaque fichier renvoient un objet `Transform`, celui-ci étant transformé en image avant d'être récupéré dans le module principal.

`NonRigid.py` regroupe les recalages B-Spline et Demons dit non rigides.

Méthode elastix_registration

Cette méthode fait appel à `Elastix Logic`, un objet tiré de l'extension Slicer Elastix. La méthode de recalage `RegisterVolumes` prend en paramètres l'image fixe, l'image à recaler, un *preset* de Slicer Elastix sélectionné au préalable, et un volume de sortie. Slicer Elastix n'est pas paramétrable par l'utilisateur autrement qu'avec les presets. Slicer Elastix est un choix d'intégration du client.

Le lancement d'un recalage exécute soit un script Python en tâche de fond, soit Slicer Elastix en tâche de fond. Lorsque le recalage se termine, un objet dit "observer" notifie l'extension, appelle la méthode définitive qui affiche l'image recalée si pas d'erreur, et arrête la barre de progression.

Note sur Slicer Parallel Processing

L'extension suivante est une réponse à un besoin qui s'est vite fait ressentir lors du développement du recalage. Certains processus comme le non rigide peuvent prendre plusieurs minutes à s'exécuter, il est hors de question de figer l'interface utilisateur lors de ces phases lourdes en calcul. Slicer Parallel Processing offre la possibilité de lancer des scripts python externes en arrière plan. Il existe dans le code une classe `RegistrationProcess` qui est utilisée pour préparer les données entrées avec `prepareProcessInput` et traite les données en sortie des scripts de recalage avec `UseProcessOutput`. Cette classe utilise le module Pickle de Python, servant à sérialiser/désérialiser des objets afin de les transmettre sous forme de flux d'octets à nos scripts de recalage.

Note sur Utilities.py

C'est un fichier utilitaire servant de lien entre les entrées utilisateurs et l'objet de recalage associé, `ImageRegistrationMethod`. Cet objet est calibré avec les paramètres entrés par l'utilisateur et se fait au moyen des méthodes du dit fichier.

Recalage typique

Il se traduit par la Figure 23. Il existe quatre types de recalage avec SimpleITK (Rigid, Affine, Non Rigid, Demons), sans compter les variations de Demons. A ceux-là s'ajoutent les *presets* Elastix, cela étant le nombre de possibilités à plus d'une vingtaine de recalage possible. Comme dit plus haut, le recalage n'a besoin que de deux images, et d'un dictionnaire distinct à notre code. Ce dictionnaire est étudié ci-dessous avec la description de chaque composant utilisateur qu'il contient.

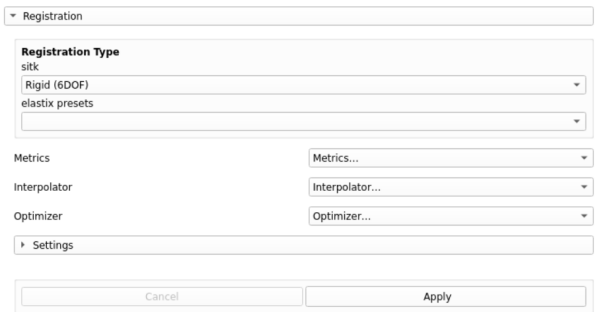

FIGURE 23 – Aperçu de l'UI de recalage.

Métriques

Ce sont des mesures de similarité statistiques qui partent du postulat qu'il existe un lien entre les intensités des deux images à recaler. L'extension en implémente quatre.

— **Mean Squares** Elle est utilisée pour minimiser la différence entre les intensités des voxels des images enregistrées.

— **Mattes Mutual Information** Elle utilise l'information mutuelle pour mesurer la similarité entre les images. L'information mutuelle mesure la quantité d'informations partagées entre deux images. La valeur liée à la métrique est `Number of Histogram Bins`, disponible sous l'onglet *Settings*. Valeur par défaut : 50.

— **Joint Histogram Mutual Information** Similaire à la métrique ci-dessus mais utilisant une méthode de calcul basée sur les histogrammes de Parzen. La valeur liée à cette métrique est *Number of Histogram Bins*, disponible sous l'onglet *Settings*. Valeur par défaut : 20.

— **Correlation** Elle utilise le coefficient de corrélation de Pearson pour mesurer la similarité entre les images (mesure la corrélation entre les intensités des voxels des deux images).

Interpolateurs

Les interpolateurs (voir Figure 24) sont des fonctions d'image qui permettent de calculer les valeurs des voxels de l'image transformée à partir des valeurs des voxels de l'image originale. Quatre versions sont disponibles dans le module.

— **Linear** Interpolation linéaire.

— **Nearest Neighbor** Interpolation par la méthode des plus proches voisins. Cela consiste à attribuer à chaque point de la grille de destination la valeur du point le plus proche de la grille source.

— **B-Spline (1, 2, 3)** Interpolation via une fonction de base B-Spline, pour interpoler les valeurs des voxels de l'image transformée à partir des valeurs des voxels de l'image originale.

— **Gaussian** Interpolation via une fonction Gaussienne.

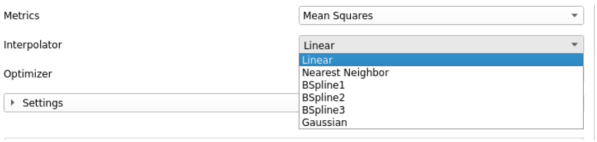

FIGURE 24 – Interpolateurs disponibles.

Optimiseurs

Ce sont des algorithmes qui permettent de trouver les paramètres optimaux pour une fonction de coût donnée. Ils sont utilisés pour trouver les paramètres de transformation qui minimisent la différence entre deux images.

— **Gradient Descent** Utilise le gradient de la métrique pour trouver le minimum local, voir la Figure 25. Les paramètres sont les suivants.

  — *Learning Rate* Paramètre qui contrôle la taille du pas de descente de gradient à chaque itération. Valeur par défaut : 0.1

  — *Number of Iterations* Valeur par défaut : 100.

— *Convergence Minimum Value* Valeur qui indique à l'algorithme quand il doit s'arrêter. Valeur par défaut : 1e-6.
— *Convergence Window Size* Nombre d'itérations pour lesquelles l'algorithme doit converger avant d'arrêter. Valeur par défaut : 10.

— **Exhaustive** : Méthode qui consiste à énumérer toutes les combinaisons possibles de paramètres pour trouver la meilleure solution. Les paramètres sont les suivants.
— *Number of Steps* Nombre de pas pour chaque dimension de paramètre. Correspond aux paramètres d'une transformation Euler 3D. C'est un vecteur de 6 valeurs (angle x, y, z, translation x, y, z). Valeur par défaut : 1, 1, 1, 0, 0, 0.
— *Step Length* Longueur de pas pour chaque dimension de paramètre. Valeur par défaut : π.
— *Optimizer Scale* Permet de spécifier les échelles des paramètres dans l'optimiseur utilisé pour minimiser la fonction de coût. C'est un vecteur d'entier. Valeur par défaut : 1, 1, 1, 1, 1, 1.

— **LBFGSB** : Algorithme d'optimisation quasi-Newtonien qui utilise une approximation de la matrice Hessienne pour trouver le minimum local. Il est plus robuste, plus rapide et efficace qu'une descente de gradient.
— *Gradient Convergence Tolerance* Seuil pour laquelle l'algorithme considère que le gradient a convergé vers un minimum local. Valeur par défaut : 1e-5.
— *Number of Iterations* Valeur par défaut : 100.
— *Maximum Number of Corrections* Nombre maximum de corrections à appliquer avant d'arrêter l'algorithme. Une correction est une étape de mise à jour des paramètres de l'optimiseur. Valeur par défaut : 5.
— *Maximum Number of Function Evaluation* Valeur par défaut : 1000.

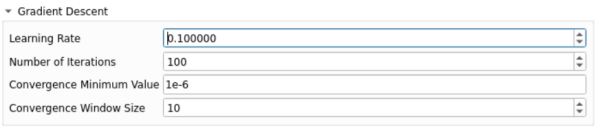

FIGURE 25 – Exemple des paramètres de la descente de gradient.

Recalage non-rigide

Les méthodes de recalages non rigides (B-Spline et Demons), ont des paramètres inhérent qui vont être détaillés ci-dessous.

— **B-Spline** Méthode de recalage déformable qui utilise des courbes B-Spline pour définir un champ de déformation continu qui fait correspondre chaque voxel d'une image en mouvement à un voxel correspondant dans une image fixe ou de référence. Ce recalage supporte une approche multi-résolution d'ordre 3 qui effectue d'abord l'enregistrement à une résolution inférieure avec moins de paramètres au premier niveau, puis adapte ou rééchantillonne les points de contrôle B-Spline à une résolution supérieure aux niveaux suivants. Voici les paramètres utilisés.
— *Transform Domain Mesh Size* Nombre de points de contrôle utilisés pour définir la transformation B-Spline. Plus il est grand, plus le recalage sera précis mais long. Valeur par défaut : 2.
— *Scale Factor* Utilisé pour mettre à l'échelle la grille de points de contrôle B-Spline à chaque niveau de résolution. Valeur par défaut : [1, 2, 4]. (Il ne peut pas être utilisé avec l'optimiseur LBFGSB, et devient inactif si sélectionné.)
— *Shrink Factor* Utilisé pour déterminer la taille de l'image à chaque niveau de résolution. Valeur par défaut : [4, 2, 1].

— *Smoothing Sigmas* Utilisés pour lisser les images à chaque niveau de résolution. Valeur par défaut : [2, 1, 0].

— **Demons** Méthode de recalage non paramétrique qui estime le champ de déformation entre deux images en minimisant une fonction d'énergie qui mesure la différence entre les deux images. L'algorithme met à jour de manière itérative un champ de déformation qui fait correspondre une image à une autre jusqu'à convergence. Notre extension propose quatre variations de Demons. *Original*, *Diffeomorphic*, *Symmetric Forces*, et *Fast Symmetric Forces*. Voici leurs paramètres.
  — *Number of Iterations* Valeur par défaut : 20.
  — *Standard Deviation* Utilisé pour contrôler l'ampleur du lissage appliqué au champ de déplacement. Valeur par défaut : 1.

Autres paramètres

Deux autres paramètres que l'on trouve dans les recalages non rigide, rigide et affines viennent compléter la gamme d'entrées du dictionnaire.

— *Sampling Strategy* Spécifie comment les échantillons sont tirés du domaine d'image fixe pour calculer la valeur de la métrique. La stratégie *None* utilise tous les voxels. *Regular* échantillonne chaque n-ième voxel en parcourant l'image dans l'ordre des lignes de balayage. Et *Random* échantillonne chaque n-ième voxel tout en parcourant l'image dans l'ordre de la ligne de balayage, puis dans chaque voxel, perturbé aléatoirement à partir du centre. Valeur par défaut : *Random*.

— *Sampling Percentage* Définit le pourcentage de voxels échantillonnés pour l'évaluation métrique. Valeur par défaut : 0.01.

5.3.2 Limites

La Figure 26 illustre à quoi ressemble l'interface utilisateur après sélection de quelques paramètres qui seront exportés dans le dictionnaire pour ensuite être utilisés dans le script `Rigid.py`. Sous cette forme minimal, l'interface utilisateur ne permet pas une maîtrise totale des algorithmes de recalage. Pour être exhaustif, beaucoup de paramètres auraient dû être rajoutés, mais dans un souci de clarté et de simplicité, seuls certains paramètres ont étés retenus. De plus, il faut être attentif aux paramètres lors d'un recalage non rigide avec B-Spline, en effet, si on utilise un *Transform Domain Mesh Size* élevé avec un vecteur *Scale Factor* conséquent, le recalage peut prendre énormément de temps. C'est à l'utilisateur d'ajuster chaque paramètre pour faire en sorte d'obtenir les résultats sous ses propres exigences (durée du recalage, précision). C'est pour cela que plusieurs options sont ouverts à l'utilisateur, selon ses besoins. Aussi, le recalage Demons nécessite d'avoir au préalable rééchantillonner l'image à recaler, il se peut que une erreur s'affiche, dans ce cas, il faut suivre les indications du message d'erreur. Enfin, il n'a pas toujours été possible de rediriger les sorties écrites vers le script principal. En l'état il affiche dans la console python les conditions d'arrêt uniquement pour les recalages rigide, B-Spline et affine. Le sortie écrite de Slicer Elastix n'a pas été implémenté, ni la sortie pour Demons qui n'utilise pas le même objet pour exécuter le recalage.

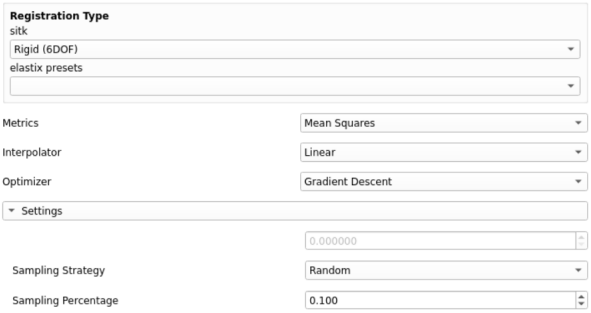

FIGURE 26 – Cas utilisateur d'un recalage rigide avec divers paramètres sélectionnés.

5.4 Carte de différence

La carte de différence a pour but d'évaluer l'erreur entre deux images. Elle apporte une estimation visuelle et permet d'évaluer les performances, avantages et désavantages des différents traitements et recalages effectués, en identifiant également les zones les plus impactées.

Avec différentes techniques, il est possible de calculer une carte de différence en ayant en entrée deux images, dans notre cas l'image initiale et l'image recalée, puis d'améliorer sa visualisation.

Les bibliothèques *VTK* et *NumPy* ont été utilisées. Elles sont déjà présentes et utilisées dans 3D Slicer. Les images sont stockées dans 3D Slicer comme des volumes *VTK*. Pour calculer la carte de différence efficacement, les deux images sont converties en tableaux *NumPy*. *NumPy* permet alors d'accéder à plusieurs fonctions optimisées.

Pour calculer une carte de différence via l'UI dédiée, il suffit de choisir un algorithme entre la valeur absolue de la différence et la différence de gradient, et éventuellement une taille de patch pour appliquer un filtre moyenneur et limiter ainsi le bruit.

La carte de différence est calculée entre l'image d'entrée et l'image cible (sélectionnées préalablement dans l'interface supérieure). Une fois le calcul terminé, la carte de différence est affichée dans la vue dédiée, c'est-à-dire la vue jaune (en bas à gauche).

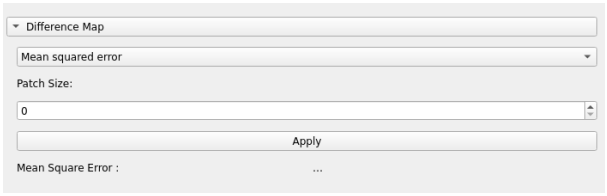

FIGURE 27 – Aperçu de l'UI de la carte de différence.

5.4.1 Valeur absolue de la différence

Soient A et B respectivement les images d'entrée et cible sélectionnées, et C la carte de différence. $C(i) = ||A(i) - B(i)||$, où i est l'index d'un voxel dans A et B.

L'avantage d'utiliser la valeur absolue dans le calcul de la carte de différence est qu'elle permet d'obtenir des résultats positifs qui reflètent l'amplitude de l'erreur de recalage. En effet, la valeur absolue ignore le signe des différences, ce qui permet de prendre en compte l'intensité des erreurs dans toutes les directions. Ainsi, si un voxel est mal aligné dans une certaine direction, la carte de différence va montrer une différence significative en termes d'amplitude, indépendamment de la direction de l'erreur.

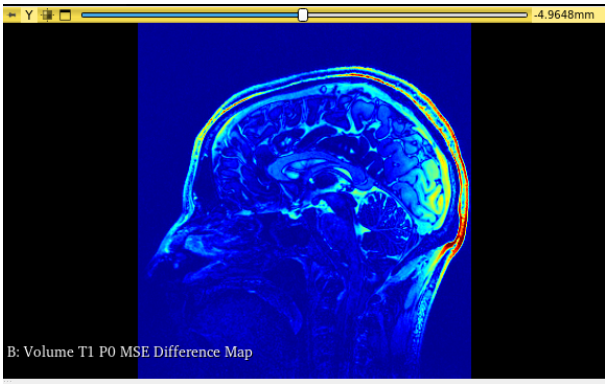

FIGURE 28 – Exemple de carte de différence en valeur absolue.

5.4.2 Différence de gradient

Soient A et B respectivement les images d'entrée et cible sélectionnées, et C la carte de différence. $C(v) = \nabla A(v) - \nabla B(v)$, où $\nabla A(v)$ et $\nabla B(v)$ sont les gradients de A et B respectivement, et v est un voxel de A et B. Le gradient de l'image est calculé grâce aux bibliothèques *VTK* et *NumPy*.

L'utilisation des gradients peut être pertinente car elle permet de quantifier la différence entre deux images en se basant sur les variations locales de l'intensité des voxels plutôt que sur leurs valeurs absolues. Cela peut être plus précis pour détecter les zones de l'image où le recalage a le plus d'impact, car les gradients mesurent les changements de l'intensité des voxels, ce qui est particulièrement pertinent pour les contours et les détails fins.

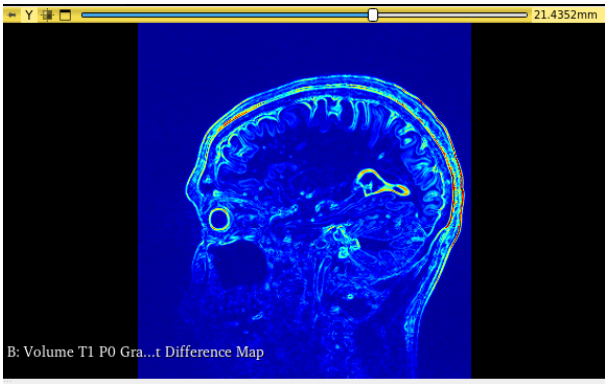

FIGURE 29 – Exemple de carte de différence de gradient.

5.4.3 Filtre moyenneur avec noyau de convolution

Ce filtre permet de flouter l'image et d'avoir une meilleur vue d'ensemble de la carte d'erreur et d'éliminer le bruit potentiel notamment sur la carte de différence de gradient. Il est applicable par l'utilisateur en entrant une taille de patch. Notre patch parcourt l'image pour moyenner la valeur du voxel et de ses voisins à l'aide de la bibliothèque *SciPy* et sa méthode `convolve`.

5.4.4 Look Up Table

La bibliothèque *VTK* permet également d'appliquer une Look Up Table (LUT) pour améliorer la visualisation de la carte de différence. Cela permet de modifier la couleur de la carte de différence pour

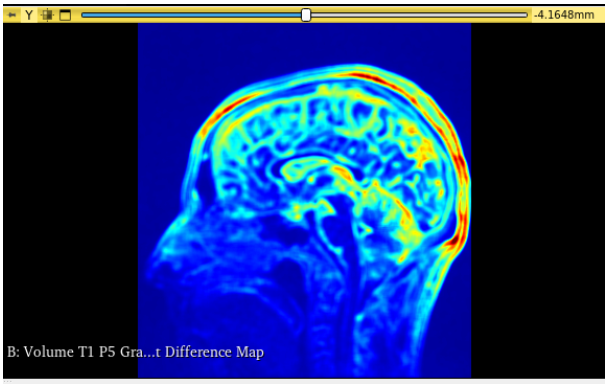

FIGURE 30 – Exemple d'un flou avec le filtre moyenneur ayant un patch de taille 5

mettre en évidence certaines zones d'intérêt ou pour améliorer la lisibilité. Nous avons utilisé la LUT *ColdToHotRainbow*, disponible dans 3D Slicer.

5.5 Plugins

Les plugins sont un moyen simple, rapide et efficace d'ajouter de nouvelles fonctionnalités au module. Un plugin se caractérise par deux fichiers : une interface graphique XML (`.ui`) et un script Python (`.py`).

L'interface graphique XML doit contenir tous les widgets que l'utilisateur veut manipuler pour paramétrer son algorithme. Le fichier `.ui` peut être rédigé manuellement ou plus simplement généré à l'aide d'outils dédiés tels que le concepteur prédéfini de 3D Slicer ou Qt Designer (inclus dans Qt Creator).

Le script Python correspondant à l'algorithme, ou à un point d'entrée aux algorithmes souhaités dans le module d'extension. Il est donc tout à fait possible d'utiliser ce script pour intégrer un algorithme personnalisé en Python ou tout autre langage, voire même pour appeler directement des algorithmes de bibliothèques. Le script doit contenir une fonction `run`, qui reçoit quatre paramètres.

- `ui` Objet représentant l'interface graphique du plugin. Il permet d'accéder aux éléments de l'UI pour les modifier ou récupérer des paramètres. Ces éléments sont accessibles en utilisant `ui.findChild(type, name)`.
- `scene` Scène de 3D Slicer (pour ajouter, supprimer ou modifier des volumes et d'autres nœuds).
- `input_volume` Image d'entrée définie dans le module. Si aucun volume n'est sélectionné en entrée, sa valeur sera None.
- `target_volume` Image cible définie dans le module. Si aucun volume n'est sélectionné en entrée, sa valeur sera None.

Pour importer un plugin, une dernière section est présente dans le panneaux de traitement, *Plugins*. Cette section est composée seulement d'un bouton *Load Plugin* qui permet d'importer un plugin (Figure 31).

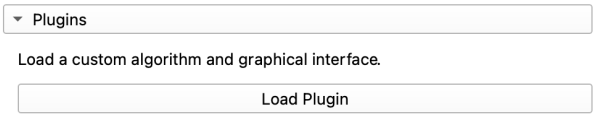

FIGURE 31 – Aperçu de l'UI d'importation de plugins.

Lors du clic sur le bouton, une fenêtre de dialogue apparaît contenant trois champs. Le champ du nom du plugin, ainsi que deux boutons pour charger une interface graphique XML et un script Python

comme indiqué sur la Figure 32. Le clic sur chacun des deux boutons affiche respectivement une fenêtre de chargement de fichier limitée aux fichiers `.ui` et `.py`. Une fois un fichier choisi, son nom est affiché à gauche du bouton correspondant.

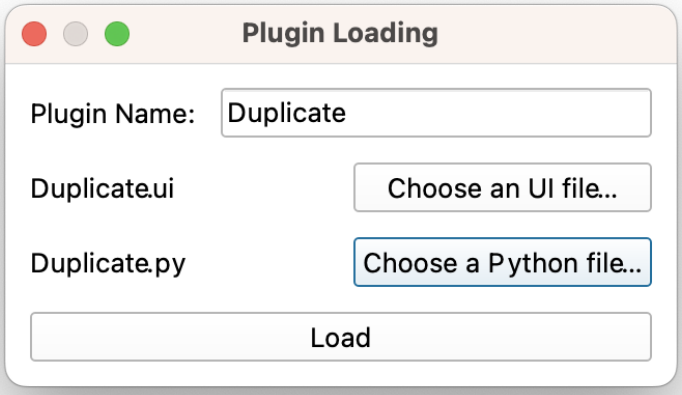

FIGURE 32 – Fenêtre de dialogue de chargement d'un plugin.

Enfin, lorsque le bouton `Load` est cliqué, la fenêtre de chargement de plugin se ferme. L'UI chargée est alors intégrée dans le panneau de traitements dans une sous-section dédiée définie par le nom du plugin, et un bouton `Run <Plugin Name>` est inséré afin de pouvoir faire appel au script chargé à tout moment. Cette intégration est représentée par la Figure 33.

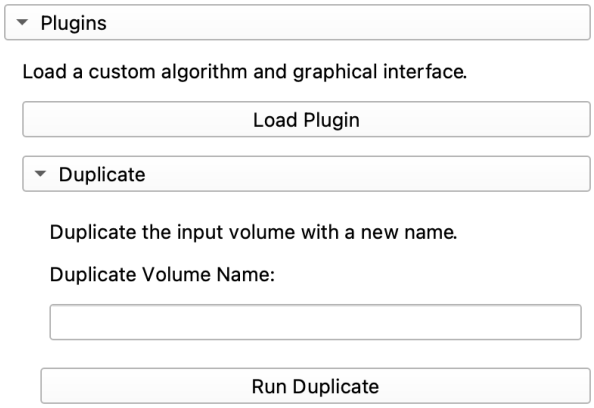

FIGURE 33 – Fenêtre de dialogue de chargement d'un plugin.

L'importation de plugins est entièrement gérée par la méthode `Widget.setup_plugin_loading`. Cette méthode est très simple. Elle se charge d'initialiser le chargement de plugins une fois au démarrage par la méthode `Widget.setup` générale et c'est terminé, il n'y a plus besoin de recharger l'outil d'importation de plugins.

Tout d'abord, un dictionnaire est défini représentant la liste de plugins, chaque nom étant associé au chemin d'accès du script Python. Celui de l'interface graphique XML n'est pas nécessaire car celle-ci est chargée directement lors de l'importation d'un plugin, alors que le script peut être appelé à tout moment

lors de l'utilisation du module. Ensuite, le bouton de chargement de plugin est alors récupéré et lié à la sous-fonction d'importation décrite ci-après.

La sous-fonction d'importation `on_plugin_loading_button_clicked` se charge de créer la fenêtre de dialogue Qt permettant le chargement d'un plugin, représentée en Figure 32. Les boutons détaillés plus tôt sont donc liés à des actions spécifiques permettant d'ouvrir une fenêtre de chargement de fichier grâce à Qt (sous-fonctions spécifiques `on_ui_file_button_clicked` et `on_python_file_button_clicked`). Les chemins des fichiers ainsi récupérés sont gardés en mémoire temporairement. Le bouton de *Load Plugin* est quant à lui lié à une autre sous fonction nommée `on_load_button_clicked` et décrite ci-après.

Enfin, la sous-fonction `on_load_button_clicked` permet quant elle de charger effectivement le plugin. Tout d'abord, le nom du plugin est vérifié car il ne peut y avoir plusieurs plugins de même nom, ainsi que la présence des fichiers requis. On peut alors charger l'UI depuis le fichier dédié grâce à la méthode de 3D Slicer `util.loadUI`, en l'insérant dans une nouvelle section du panneau de traitements nommé en accord avec le plugin.

On ajoute alors le bouton `Run <Plugin Name>` mentionné précédemment en le liant à l'appel de la fonction `run` du script Python qui lui est alors stocké dans le dictionnaire des plugins.

6 Conclusion

Ce projet est achevé mais il existe de nombreuses possibilités d'améliorer cette proposition. Les User Stories ont toutes été traitées, nous offrons donc ici une solution efficace aux problèmes de dispersion et de complexité des différentes fonctionnalités implémentées dans 3D Slicer concernant les prétraitements et le recalage, à laquelle s'ajoute une automatisation de certains traitements, une possibilité de comparaison via la carte de différence, et surtout l'ouverture vers le monde des possibles avec les plugins.

6.1 Bilan

Tout d'abord, notre extension répond en effet aux critères d'acceptabilité de la US I concernant les prétraitements. Conformément à la maquette mise en place (Figure 2), l'UI créée offre bien la possibilité de sélectionner une image à traiter et une image de référence grâce à des listes déroulantes (Figure 14), et les affiche dans la disposition à droite, chacune dans une vue dédiée. De plus, notre extension comporte les trois prétraitements prévus et fonctionnels, à savoir la sélection de ROI par seuillage, le rognage manuel et automatique, ainsi que le rééchantillonnage. De manière similaire aux maquettes de paramétrage des prétraitements (Figures 3, 4 et 5), ceux-ci sont en effet paramétrables (Figures 15, 19, 20, et 22). Finalement, lorsque pertinent, une prévisualisation du prétraitement à venir ou une visualisation finale peuvent s'afficher à la guise de l'utilisateur (Figures 16, 17 et 21).

Les critères d'acceptabilité concernant la US II sur le recalage sont respectés, la section recalage est effective, fonctionnelle et a été testée minutieusement. Les utilisateurs finaux pourront recaler les images d'entrée et cible avec des paramètres Prédéfinis ou entrés par celui-ci, l'image recalée s'affichera dans la vue prévu à cet effet. En plus ce des critères, des ajouts bénéfiques et conviviaux pour les utilisateurs ne sont pas à négliger ; l'activation du processus s'exécute en arrière plan, ainsi il n'est pas bloquant, en outre un bouton *Cancel* est à la disposition de l'utilisateur si celui-ci trouve que le recalage prend trop de temps.

La carte de différence répond à toutes les exigences induite par la US III, c'est-à-dire deux algorithmes différents permettant d'évaluer les endroits où le recalage a été le plus ou le moins efficace. La LUT vient appuyer sur les zones les plus impactées avec un gradient de couleur instinctif et l'opération noyau de convolution est efficace pour limiter le bruit, notamment sur les cartes de gradient.

Enfin, les critères d'acceptabilité de la US IV sont également respectés. Il est en effet possible de charger des plugins, définis par une interface graphique XML et un script Python. Ces plugins sont intégrés au panneau de traitement, et permettent une manipulation de la scène, mais aussi de l'architecture et des données de notre module d'extension, notamment les images d'entrée et cible.

6.2 Perspectives

Au niveau des prétraitements, les perspectives d'améliorations sont nombreuses. Nous nous sommes effectivement concentrés sur des traitements généraux avec des paramètres minimaux. Mais tous ces prétraitements pourraient également posséder un mode "avancé" comprenant des paramètres plus spécifiques, ou simplement manuels, afin d'affiner l'utilisation selon les besoins. Ceci étant dit, il est tout de même possible de faire des traitements avancés en utilisant des plugins personnalisés.

En particulier, la sélection de ROI pourrait être grandement améliorée avec une segmentation et non pas un seuillage binaire. L'utilisateur pourrait segmenter son image et alors choisir le(s) tissu(s) qu'il vise à sélectionner plus précisément.

Concernant le rééchantillonnage, il pourrait être fait d'une manière supplémentaire, par exemple en rééchantillonnant les deux images sélectionnées vers une nouvelle résolution spatiale définie, autre que les deux déjà existantes. Ou encore, on pourrait également choisir un mode d'interpolation différent pour obtenir des voxels de valeurs différentes selon le cas. On rentre alors dans les paramétrisations avancées évoquées précédemment.

La prévisualisation de manière générale n'est pas encore optimisée. Que ce soit pour la sélection de ROI, le rognage manuel, ou le rognage automatique, le traitement est fait même si l'utilisateur ne le confirme jamais, afin de créer l'objet de prévisualisation. Une perspective d'optimisation viserait donc à créer l'objet de prévisualisation, sans avoir à effectuer le traitement dans tous les cas.

De plus, concernant le rognage, on pourrait ajouter la possibilité d'interagir avec les points de contrôle, afin de permettre à l'utilisateur de définir encore plus précisément la zone à rogner.

Dans la partie dédiée au recalage le prochain jalon de développement aurait été la possibilité d'exécuter deux recalages sur une même image : l'un à l'intérieur de la ROI et l'autre à l'extérieur, afin d'assurer lors du recalage une prise en compte des ROI.

Notre extension pourrait également être améliorée pour permettre à l'utilisateur de comparer ses propres algorithmes de recalage en implémentant des métriques de performances entre les recalages en fonction de l'utilisation : rapidité, précision, polyvalence.

Enfin, on pourrait également se concentrer sur l'ajout d'autres bibliothèques de recalage plus spécifique aux images cérébrales comme BRAINS déjà disponible dans 3D Slicer.

La carte de différence aurait pu être amélioré de plusieurs façons. La première aurait été de mettre un chargement car l'opération du noyau de convolution peut prendre du temps, et par ailleurs paralléliser cette opération. Une autre aurait été de pouvoir choisir sa LUT, ça aurait augmenté l'accessibilité pour les personnes daltoniennes par exemple. Enfin devoir appuyer sur le bouton *Apply* à chaque modification peut casser l'ergonomie, une carte de différence automatique aurait pu être mieux mais demande une meilleur gestion de la mémoire.

L'importation d'algorithmes via les plugins pourrait également être améliorée afin de faire tourner en arrière plan les scripts, à l'aide de l'extension *Slicer Parallel Processing*, ou en chargeant automatiquement les scripts présents dans un certain répertoire au lancement du module pour éviter le chargement manuel.

6.3 Notes de fin

La fin des réunions s'est conclue sur une satisfaction des deux parties, le client satisfait par la solution, et nous, développeurs sommes satisfaits et fiers d'avoir mené un projet qui sera utilisé voire amélioré. Nous avons apporté notre pierre à l'édifice qu'est 3D Slicer, et nous sommes convaincus que notre développement extensible sera reconnu comme pratique pour laisser place à de nombreuses évolutions.

Références

B. B. Avants, C. L. Epstein, M. Grossman, and J. C. Gee. Symmetric Diffeomorphic Image Registration with Cross-Correlation : Evaluating Automated Labeling of Elderly and Neurodegenerative Brain. *Medical Image Analysis*, 12(1) :26–41, 2008. doi : 10.1016/j.media.2007.06.004.

N. Betrouni. Le recalage en imagerie médicale : de la conception à la validation. *IRBM*, 30(2) :60–71, 2009. doi : 10.1016/j.irbm.2008.12.003.

A. Fedorov, R. Beichel, J. Kalpathy-Cramer, J. Finet, J.-C. Fillion-Robin, S. Pujol, C. Bauer, D. Jennings, F. Fennessy, M. Sonka, J. Buatti, S. Aylward, J. V. Miller, S. Pieper, and R. Kikinis. 3D Slicer as an image computing platform for the Quantitative Imaging Network. *Magnetic Resonance Imaging*, 30 (9) :1323–1341, 2012. doi : 10.1016/j.mri.2012.05.001.

S. Klein, M. Staring, K. Murphy, M. A. Viergever, and J. P. W. Pluim. Elastix : A Toolbox for Intensity-Based Medical Image Registration. *IEEE Transactions on Medical Imaging*, 29(1) :196–205, 2010. doi : 10.1109/TMI.2009.2035616.

B. Lowekamp, D. Chen, L. Ib'a nez, and D. Blezek. The Design of SimpleITK. *Frontiers in Neuroinformatics*, 7 :45, 2013. doi : 10.3389/fninf.2013.00045.

K. Marstal, F. Berendsen, M. Staring, and S. Klein. SimpleElastix : A User-Friendly, Multi-lingual Library for Medical Image Registration. *Journal of Open Source Software*, 1(7) :100, 2016. doi : 10.21105/joss.00100.

T. S. Yoo, M. J. Ackerman, W. E. Lorensen, W. Schroeder, V. Chalana, S. Aylward, D. Metaxas, and R. Whitaker. Engineering and algorithm design for an image processing API : a technical report on ITK – the Insight Toolkit. *Studies in Health Technology and Informatics*, 85 :586–592, 2002.

7 Annexes

User Story	Critères d'acceptabilité
En tant qu'**utilisateur**, je dois pouvoir…	
I. Appliquer des prétraitements (sélection de ROI par seuillage, rognage manuel ou automatique et rééchantillonnage automatique) avec des paramètres minimaux, à une image d'entrée, en fonction d'une image de référence, via une interface dédiée les regroupant, en affichant les deux images ainsi qu'une prévisualisation en temps réel.	1. L'interface de prétraitements doit permettre de sélectionner l'image d'entrée et l'image de référence lorsque c'est nécessaire, et de les afficher, tout comme l'image résultante. 2. Les options de prétraitement doivent inclure la sélection de ROI par seuillage, le rognage manuel ou automatique, et le rééchantillonnage automatique. 3. Les options de prétraitement doivent être paramétrables via l'interface dédiée. 4. L'interface doit afficher une prévisualisation du traitement souhaité, en temps réel.
II. Appliquer des algorithmes de recalage rigide ou non-rigide, paramétrables manuellement ou à l'aide de presets, via une interface dédiée, en affichant l'image à recaler et l'image de référence, puis l'image recalée, dans la vue.	1. L'interface de recalage doit permettre de sélectionner l'image à recaler et l'image de référence. 2. Les options de recalage doivent inclure des algorithmes rigides et non rigides. 3. Les options de recalage doivent être paramétrables manuellement ou à l'aide de presets. 4. L'interface doit afficher une vue de l'image à recaler, l'image de référence, ainsi que l'image recalée.
III. Calculer une carte de différence entre deux images choisies, voxel-à-voxel ou par gradients et avec possibilité d'utiliser des patchs, via une interface dédiée, et de l'afficher dans la vue.	1. L'interface de calcul de la carte de différence doit permettre de sélectionner les images à comparer. 2. Les options de calcul de la carte de différence doivent inclure la possibilité de calculer la différence pixel-à-pixel ou par gradients, et de spécifier l'utilisation de patchs. 3. L'interface doit afficher la carte de différence calculée dans la vue.
En tant que **développeur,** je dois pouvoir…	
IV. Importer une interface XML et un script Python à utiliser dans le cadre d'un plugin.	1. L'interface de développement doit permettre d'importer une interface XML et un script Python pour une utilisation en tant que plugin. 2. Les plugins doivent être intégrables dans le logiciel principal et accessibles via une interface dédiée. 3. Les plugins doivent être compatibles avec les autres fonctionnalités du logiciel, en s'intégrant au sein de l'architecture de l'extension.

FIGURE 34 – Tableau récapitulatif des User Stories et leurs critères d'acceptabilité respectifs.