Rapport Miniprojet ITK/VTK

ernest.bardon, ewan.lemonnier, axelle.mandy

`https://github.com/Sevenudeloin/VITK-turbo-lamp`

1 Recalage d'images

Le recalage d'images est une problématique qui peut être décomposée en plusieurs sous-parties[1].

D'une part cela implique une image cible et une image qui va être transformée pour s'y superposer au mieux. Ici, notre image cible est *case6_gre1.nrrd* tandis que l'image à transformer est *case6_gre2.nrrd*.

Ensuite il faut choisir une transformation qui correspond à la fonction de transition de l'image à transformer vers l'image cible.

Enfin, afin d'évaluer les performances de la transformation, on utilise des métriques qui vont servir à déterminer les meilleurs paramètres pour la transformation à l'aide de l'optimiseur.

1.1 Choix des métriques

Nous avons sélectionné deux métriques différentes afin d'évaluer la performance de nos méthodes de recalage :

1. **Mean Squares Metric** :

Cette métrique repose sur une différence pixel à pixel et nécessite que les deux images soient issues de la même modalité. En étudiant les images, nous avons conclu que l'une comme l'autre avaient été acquises avec la séquence FLAIR. De ce fait, la condition nécessaire à l'utilisation de la métrique MSE est remplie.

2. **Mattes Mutual Information Metric** :

Bien que l'acquisition n'a pas été multimodale, on ne peut exclure un éventuel changement de machine entre les deux IRM. De ce fait, il nous a paru cohérent d'utiliser cette métrique en plus du MSE qui est justement sensible aux changements d'intensité.

1.2 Choix de l'optimiseur

Le choix de l'optimiseur s'est fait empiriquement en prenant en compte le temps d'exécution ainsi que le résultat final du recalage.

Il est important de préciser que malgré plusieurs tentatives nous n'avons pas réussi à mettre en place un système de contrôle (avec le design pattern de l'observeur) pour suivre les différentes itérations des optimisateurs. Ainsi, nous n'avons pas été en mesure d'explorer et d'améliorer les paramètres des optimiseurs aussi bien que prévu.

1. **LBFGSB** :

Nous avons commencé par plusieurs tentatives en utilisant une transformation BSpline et un optimiseur LBFGSB (comme on trouve dans l'exemple[2]). Cet optimiseur est en effet approprié au nombre de paramètres de la transformation BSpline. Toutefois les résultats du recalage étaient très mauvais et en essayant de changer les paramètres, le temps d'exécution dépassait les vingt minutes.

2. **RegularStepGradientDescentOptimizerv4** :

Nous avons décidé de simplifier notre pipeline avec des transformations plus basiques et une descente de gradient à pas constant. Les résultats étaient satisfaisants visuellement et le recalage ne prenait plus qu'une minute, c'est donc le RegularStepGradientDescentOptimizerv4 que nous avons choisi pour notre pipeline finale. Nous avons utilisé les paramètres suivants :

- MaximumStepLength = 0.2
- MinimumStepLength = 0.0001
- NumberOfIterations = 200

Ces paramètres ont été choisi de façon arbitraire en essayant de choisir des valeurs "moyennes" pour avoir une convergence suffisamment rapide tout en restant efficace.

1.3 Choix de la méthode de transformation

Il existe une grande variété de transformations disponibles avec ITK. De manière générale, on peut les catégoriser de la façon suivante :

- **Translation / Rigid** : Pour corriger de petits désalignements (ex: structures rigides)
- **Affine / Similarity** : Pour les images avec de petites déformations ou différentes échelles (ex: acquisitions à des moments différents)
- **BSpline / Thin Plate Spline / Elastic / Demons** : Pour les déformations plus complexes (ex: tissus mous)

Puisque nous travaillons avec des images de cerveaux les deux premiers groupes de transformations sembles bien correspondre à nos besoins.

Pour les analyses suivantes nous avons utilisé un optimiseur RegularStepGradientDescentOptimizerv4 tel que décrit ci-dessus.

Fig. 1 : Metrics depending on each transform function

	Mean Squares ↓	**Mattes Mutual Information**
Translation	11180.78	-0.626
Rigid	9466.54	-0.657
Affine	58473.66	-0.314

La Mean Squares métrique est optimisée lorsqu'elle est proche de zéro tandis que la Mattes Mutual Information métrique est meilleure plus le nombre est grand (pour les nombres négatifs, un grand nombre vaut mieux qu'un petit aussi).

D'après les résultats quantitatifs, on peut constater que la transformation rigide obtient les meilleurs résultats à la fois par rapport à la MSE et par rapport à la Mattes Mutual Information. Cette première impression est soutenue par les résultats qualitatifs obtenus à l'aide de la bibliothèque `matplotlib` :

Fig. 2.1 : Translation Transform on slice 80

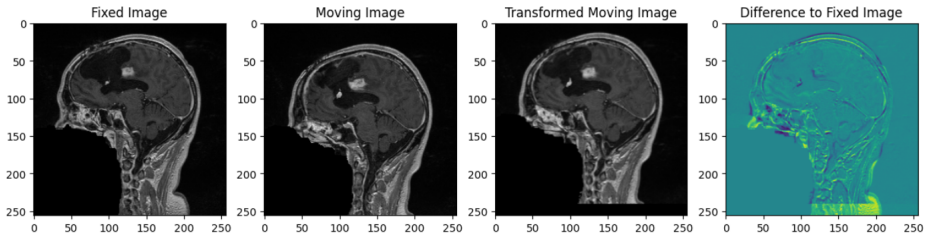


Fig. 2.2 : Rigid Transform on slice 80

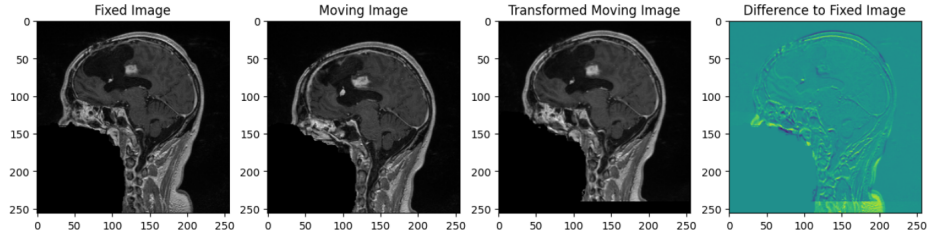
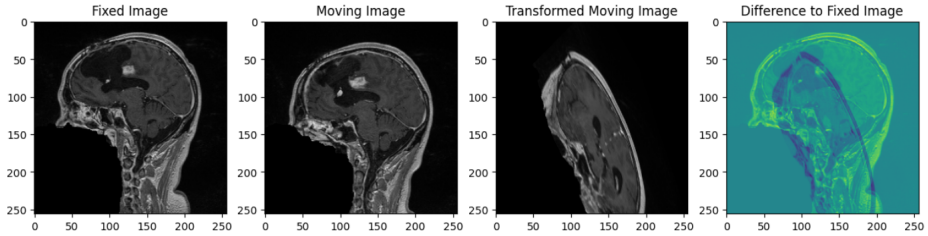


Fig. 2.3 : Affine Transform on slice 80



On y retrouve une forte similarité entre la translation et la transformation rigide, qui reste toutefois meilleure d'après la différence. La transformation affine donne de très mauvais résultats.

2 Segmentation

Nous avons testé plusieurs méthodes de segmentation afin de trouver une zone d'intérêt et ainsi sélectionner la plus adaptée.

Le projet étant en groupe, nous nous sommes mis d'accord pour un input et output pour chaque étape.

Pour plus de simplicité, nous avons choisi de sauvegarder chaque étape sous le format NRRD comme les données initiales. Il nous suffit donc de lire le fichier de l'étape précédente pour relier les étapes entre elles.

2.1 Pré-processing

Quelques étapes intermédiaires sont cependant nécessaires avant application.

2.1.1 Séparation en image 2D

Pour avoir une pleine compréhension des méthodes que nous utilisons, nous avons séparé les images de voxels en part d'image en pixels. Cela nous permet d'appliquer individuellement la segmentation sur chacune des parties de l'image.

2.1.2 Normalisation

Chaque image possède certains pixels avec des valeurs supérieures à 255 puisqu'elles sont encodées en "unsigned short". Dans ces cas-là, la segmentation pourrait échouer de plusieurs manières :

- Erreur lors du traitement
- Non prise en compte des pixels supérieurs à 255
- Prise en compte des pixels supérieurs à 255 comme des pixels de valeur 255, ce qui implique de la perte d'information

Afin d'éviter tout ces problèmes lors de la segmentation nous avons normalisé les images pour avoir uniquement des valeurs entre 0 et 255 ("unsigned char").

2.1.3 Smooth

Le résultat des scans étant très nets, les résultats de la segmentation pourraient être faussés par les micro-variations de valeur.
Pour cette raison, nous utilisons la fonction `itk.GradientAnisotropicDiffusionImageFilter` pour appliquer un lissage aux images avant la segmentation. Nous utilisons les mêmes paramètres que pendant le tp (c-à-d `NumberOfIterations=20, TimeStep=0.04, ConductanceParameter=3`).

2.2 Méthodes de segmentation

Parmi les différentes méthodes traditionnelles de segmentation, nous en avons testé trois.

2.2.1 Binarisation simple avec seuillage

Fonction : `itk.ThresholdImageFilter`

Cette méthode binarise l'image par rapport à un seuil. Tout les pixels au dessus du seuil sont blanc et les autres sont noir.
Elle est très simple d'utilisation mais elle s'avère très peu efficace pour de la segmentation d'image avancée.

2.2.2 Méthode d'Ostu

Fonction : `itk.OtsuThresholdImageFilter`

La méthode d'Otsu est utilisée pour effectuer un seuillage automatique à partir de la forme de l'histogramme de l'image.
On peut la combiner avec la fonction `itk.ConnectedComponentImageFilter` qui connecte différentes parties de l'image entre elles.

2.2.3 Propagation régionale avec seuillage

Fonction : `itk.ConnectedThresholdImageFilter`

Cette méthode est une propagation des pixels sélectionnés comme blancs en partant d'un pixel de l'image. Le pixel choisi dans l'image devient blanc, puis tous les pixels supérieur au seuil bas et connectés à un pixel blanc deviennent blancs à leur tour. On répète le processus jusqu'à ce que l'image ne change plus.

C'est cette méthode que nous avons gardé pour ce projet. Grâce au recadrage nous pouvons placer le point de départ de la fonction à des coordonnés fixes sur la région de la tumeur (x = 125 et y = 65).
Après plusieurs tests, nous avons fixé le seuil bas à 100 ce qui nous donne un résultat de segmentation convenable.

3 Visualisation

3.1 Visualisation 2D tranche par tranche

Pour la visualisation, on propose une vue en coupe sagittale[3] tranche par tranche, avec à gauche la première image et à droite la seconde. On peut voir en rouge la zone de la tumeur segmentée. En utilisant le clic gauche de la souris, on peut monter et descendre pour changer de tranche.

On fait égelement une étude globale de la tumeur, avec la taille de la tumeur en nombre de pixels et en pourcentage de volume dans le crâne. On a également le facteur d'évolution du volume de la tumeur entre les deux images.

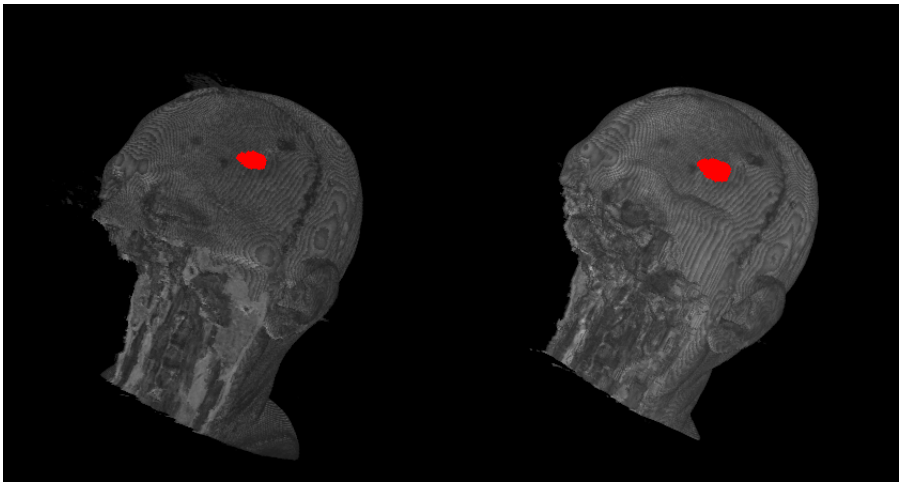
Fig. 3 : Visualisation 2D - Vue sagittale

(En utilisant cette vue, on remarque que sur les premières et dernières tranches on a des erreurs de segmentation dues au recalage. On nettoie donc après-coup le masque sur ces tranches pour avoir une vue plus propre.)

3.2 Visualisation 3D (abandonnée)

Après avoir fait plusieurs essais de visualisation 3D comme visible ci-dessous, nous avons pris le parti de rester sur la visualisation en 2D tranche par tranche. En effet, nous trouvons que cette vue permet un meilleur suivi de l'évolution de la tumeur entre les deux images.

Fig. 4 : Visualisation 3D



4 Bibliographie

- [1] https://itk.org/Doxygen413/html/RegistrationPage.html
- [2] https://itk.org/Doxygen/html/Examples_2RegistrationITKv4_2DeformableRegistration8_8cxx-example.html#_a9 - [3] https://stackoverflow.com/a/56765160

5 Répartition du travail

- Segmentation : Ernest Bardon
- Recalage et Visualisation : Ewan Lemonnier, Axelle Mandy