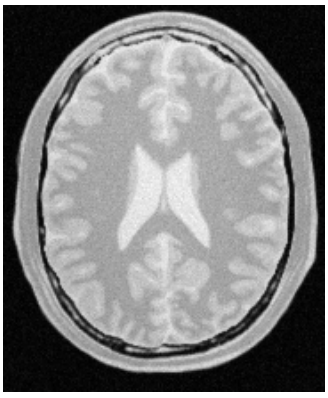
Année universitaire

2023 - 2024

Majeure IMI — Partie 3 — 5ETI

Acquisition Calibrage et Reconstruction 3D

TP de recalage



Mehdi Ayadi

Eric Van Reeth

Organisation du TP

Objectifs

Le but de ce TP est de se familiariser avec la bibliothèque ITK (Insight Toolkit) développée par Kitware, et d'utiliser les méthodes de recalage présentes dans cette bibliothèque. ITK (http://www.itk.org) est une bibliothèque de traitement d'images qui n'a aucune interface graphique. Elle est disponible à la fois en C++ et en Python. Il est conseillé de se référer à la documentation d'ITK (compatible avec la version installée sur les machines) :
https://itk.org/Doxygen412/html/classes.html
Cette documentation contient notamment l'ensemble des méthodes associées aux classes que nous allons utiliser. Il est donc important de s'y référer régulièrement.

Déroulement

Ce TP s'effectue en binôme par poste informatique (sous LINUX).
Le langage utilisé sera Python, avec la librairie suivante :

```
import itk
```

L'utilisation du format Jupyter Notebook (`.ipynb`) est fortement conseillée pour faciliter l'implémentation et le debug des exercices de manière indépendante. Veillez bien à utiliser le même environnement virtuel Python que pour les autres TP du module.

Évaluation

Ce TP n'est pas évalué, mais les concepts vus en TP seront au programme du DS.

1 Prise en main d'ITK

La librairie ITK, écrite à l'origine en C++ et étendue en Python en 2019, contient un grand nombre d'algorithmes de traitement d'image. Ces algorithmes sont appelés sous la forme générique de **filtre**, qui sont assimilables à des classes. La démarche pour traiter, déformer, ou recaler des images consiste donc à générer une instance d'un filtre, paramétrer l'instance créée puis en extraire la sortie en appelant les méthodes correspondantes.
Un exemple simple de traitement (lissage d'une image avec un filtre Gaussien) peut donc être réalisé ainsi :

```
# ----------------------
# Lecture de l'image
# ----------------------
input_filename = 'myImage.png'
image = itk.imread(input_filename)
ImageType = type(image) # On récupère le type de l'image
# ----------------------
# Création et paramétrage du filtre
# ----------------------
smoothFilter = itk.SmoothingRecursiveGaussianImageFilter[ImageType,
 ↪  ImageType].New() # On créé une instance du filtre qui prend en entrée
 ↪  et ressort des images du même type que l'image d'entrée
smoothFilter.SetInput(image) # On spécifie l'image d'entrée du filtre
```

```
smoothFilter.SetSigma(sigma) # On sélectionne la variance du filtre
# ----------------------
# Écriture de la sortie du filtre
# ----------------------
itk.imwrite(smoothFilter.GetOutput(), 'myImage_smoothed.png') # On écrit
 ↪  sur le disque la sortie du filtre Gaussien
```

1. En reprenant le code précédent, appliquer le lissage Gaussien avec plusieurs variances sur l'image *brain.png*
2. En quelle unité doit être spécifiée la variance du filtre ?
3. En vous inspirant de l'exemple précédent, écrire sur le disque une image contenant la différence entre l'image originale et l'image lissée. Utiliser le filtre `AbsoluteValueDifferenceImageFilter`, et les méthodes associées :
   `SetInput1(image1)`
   `SetInput2(image2)`

2 Exercice 2

L'objectif de cet exercice est de recaler l'image d'entrée *BrainProtonDensitySliceShifted13x17y.png* (moving image), sur l'image de référence *BrainProtonDensitySliceBorder20.png* (fixed image).
Le paramétrage du recalage inclut la spécification :

- d'une **métrique** qui mesure la similarité entre les deux images.
- d'une **transformation** qui contient les paramètres de recalage à estimer.
- d'un **interpolateur** qui indique comment calculer les valeurs de pixels lorsque la transformation est appliquée.
  On choisira ici un interpolateur linéaire : `LinearInterpolateImageFunction`
- d'un **optimiseur** qui spécifie la méthode d'optimisation utilisée pour converger vers les paramètres de transformation optimaux.
  On choisira ici une descente de gradient à pas fixe : `RegularStepGradientDescentOptimizer`

2.1 Paramétrage des caractéristiques

Chacune des 4 caractéristiques du recalage peut être paramétrée. Par exemple, il est possible de fixer le nombre maximal d'itération et de contraindre le pas de descente de l'optimiseur avec le code suivant :

```
optimizer = itk.RegularStepGradientDescentOptimizer.New() # Instance de la
 ↪  classe d'optimiseur choisie
optimizer.SetMaximumStepLength( ... ) # Borne supérieure du pas de
 ↪  descente (en pixel)
optimizer.SetMinimumStepLength( ... ) # Borne inférieure du pas de
 ↪  descente (en pixel)
optimizer.SetNumberOfIterations( ... ) # Nombre maximal d'itération
```

Il peut également être intéressant d'initialiser les paramètres de transformation grâce au code suivant :

```
initialTransform = itk.???[itk.D, 2].New() # Instance de la classe de
↪  transformation choisie
initialParameters = initialTransform.GetParameters() # Récupération des
↪  paramètres de la transformation
initialParameters[0] = ??? # Valeur du 1er paramètre
initialParameters[1] = ??? # Valeur du 2ème paramètre
... # etc...
```

2.2 Exécution du recalage

Une fois les paramètres fixés, le recalage peut être lancé. Une instance de la classe `ImageRegistrationMethod`, doit donc être créée, pour ensuite appliquer l'ensemble des caractéristiques préalablement paramétrées :

```
registration_filter = itk.ImageRegistrationMethod[fixed_image_type,
↪  moving_image_type].New() # Instance de la classe de recalage
registration_filter.SetFixedImage( ... ) # Image de référence
registration_filter.SetMovingImage( ... ) # Image à recaler
registration_filter.SetOptimizer( ... ) # Optimiseur
registration_filter.SetTransform( ... ) # Transformation
registration_filter.SetInitialTransformParameters(initialParameters) #
↪  Application de la transformation initiale
registration_filter.SetInterpolator( ... ) # Interpolateur
registration_filter.SetMetric( ... ) # Métrique
registration_filter.Update() # Exécution du recalage
```

2.3 Récupération et écriture des résultats

Après avoir exécuter le recalage, la transformation estimée peut être obtenue via :
`final_transform = registration_filter.GetTransform()`
Afin d'écrire l'image recalée sur le disque, il est nécessaire d'appliquer la transformation obtenue à l'image (moving) originale. Pour cela, il faut appliquer la tranformation à l'image d'entrée via un ré-échantillonnage.

```
resample_filter =
↪  itk.ResampleImageFilter[fixed_image_type,moving_image_type].New() #
↪  Instance de la classe de ré-échantillonnage
resample_filter.SetInput( ... ) # Image d'entrée
resample_filter.SetTransform(final_transform)
resample_filter.SetSize(fixedImage.GetLargestPossibleRegion().GetSize())
itk.imwrite(resample_filter.GetOutput(), 'image_recalee.png') # Ecriture
↪  de l'image recalée sur le disque
```

2.4 Travail à faire

1. En regardant les deux images à recaler, déduire la transformation et la métrique à utiliser pour le recalage et trouver les filtres ITK correspondants.
   Une liste (non exhaustive) des métriques et transformations disponibles avec ITK est donnée en Annexe.

2. En s'inspirant des extraits de codes ci-dessus, écrire un code permettant d'effectuer le recalage entre les deux images fournies
3. Enregistrer l'image recalée et comparer-la à l'image de référence pour vérifier que le recalage a bien fonctionné
4. Analyser la convergence du processus de recalage (voir Annexe), et afficher l'évolution des paramètres de transformation estimés ainsi que l'évolution de la métrique.
5. Écrire sur le disque l'image de différence entre l'image recalée et l'image de référence.
   L'utilisation de `RescaleIntensityImageFilter` est requise pour améliorer la visualisation.

3 Exercice 3

Nous cherchons maintenant à effectuer le recalage de l'image *BrainProtonDensitySliceR10X13Y17.png* (moving) sur l'image *BrainProtonDensitySliceBorder20.png* (fixed).

1. Quelle transformation et quelle métrique utiliser ? Trouver les filtres ITK correspondants.
2. En vous inspirant de l'exercice précédent, réaliser le recalage entre les deux images

Pensez à utiliser la fonction `SetScales()` pour prendre en compte les différences d'échelle entre les paramètres de transformation à estimer : `optimizer->SetScales(scales)`

4 Exercice 4

Nous cherchons à effectuer le recalage de l'image *BrainProtonDensitySliceR10X13Y17.png* (moving) sur l'image de référence *BrainT1SliceBorder20.png* (fixed).

1. Quelle transformation et quelle métrique utiliser ? Trouver les filtres ITK correspondants.
2. En vous inspirant des exercices précédents, réaliser le recalage entre les deux images

5 Exercice 5

Créer un algorithme qui permette de construire un panorama à partir des deux images couleurs : *PitonDeLaFournaise1.jpg* et *PitonDeLaFournaise2.jpg*.

Annexe

Transformations disponibles

Liste des transformations disponibles en ITK (consulter l'aide en ligne pour plus de détails) :

- `itk.CenteredAffineTransform[ TScalarType, NDimensions ]`
- `itk.CenteredRigid2DTransform[ TScalarType ]`
- `itk.CenteredSimilarity2DTransform[ TScalarType ]`
- `itk.Euler2DTransform[ TScalarType ]`
- `itk.Rigid2DTransform[ TScalarType ]`
- `itk.ScaleTransform[ TScalarType, NDimensions ]`
- `itk.TranslationTransform[ TScalarType, NDimensions ]`

Métriques disponibles

Liste des métriques disponibles en ITK (consulter l'aide en ligne pour plus de détails) :

- `itk.MattesMutualInformationImageToImageMetric[ TFixedImage, TMovingImage ]`
- `itk.NormalizedCorrelationImageToImageMetric[ TFixedImage, TMovingImage ]`
- `itk.MeanSquaresImageToImageMetric[ TFixedImage, TMovingImage ]`

Export de la valeur de la fonction de coût

Afin d'analyser l'évolution du processus de recalage, il est possible de spécifier une fonction qui sera appliquée à chaque itération de l'algorithme. Il suffit :

- d'ajouter un « observer » (callback) à l'optimiseur
- de spécifier la fonction à appeler
- d'écrire la fonction en question

L'exemple ci-dessous permet d'extraire la valeur de la métrique à chaque itération :

```
optimizer.AddObserver(itk.IterationEvent(), getMetricValue) # A appliquer
 ↪  avant l'exécution du recalage
f=[] # Initialisation d'une liste f dans laquelle sera stockée les valeurs
 ↪  successives de la métrique
def getMetricValue():
   f.append(optimizer.GetValue())
```