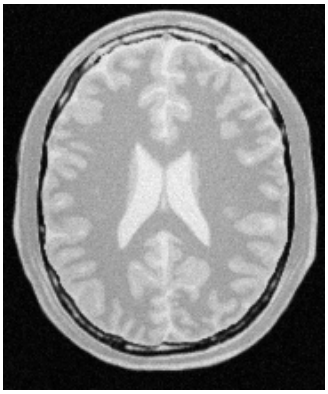
Année universitaire

2021 - 2022

Majeure IMI — Partie 3 — 5ETI

Acquisition Calibrage et Reconstruction 3D

TP de recalage



Julien Jomier

Mehdi Ayadi

Eric Van Reeth

Organisation du TP

Objectifs

Le but de ce TP est de se familiariser avec la bibliothèque ITK (Insight Toolkit) 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. Nous produirons donc des images 2D qui seront visualisées avec un logiciel de visualisation d'images générique (Gimp ou autres). 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 de l'IDE est fortement conseillée pour faciliter l'implémentation et le debug des codes. Il est également recommandé d'implémenter votre TP sous le format Jupyter Notebook, en sauvegardant votre script au format `.ipynb`. Veillez bien à utiliser `Python3`, et non pas `Python2.7`. Pour cela, veillez à ce que `VSCode` utilise l'environnement Python adéquat : `/opt/python/cpe/bin/python3`

Évaluation

Ce TP n'est pas évalué.

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 équivaut à une classe. La démarche pour traiter, déformer, ou recaler des images consiste donc à créer une instance du filtre, paramétrer l'instance créée puis en extraire la sortie en appelant les méthodes correspondantes.
Un exemple simple de traitement 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
```


Julien Jomier
Mehdi Ayadi
Eric Van Reeth

```
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 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`.

2 Recalage par translation

L'objectif de cet exercice est de recaler l'image *BrainProtonDensitySliceShifted13x17y.png* (moving image), sur l'image de référence *BrainProtonDensitySliceBorder20.png* (fixed image). Remarquons que ces deux images sont translatées l'une par rapport à l'autre. Seuls 2 paramètres sont donc ici à estimer pour recaler ces deux images : la translation selon x et selon y.
Le paramétrage du recalage inclut la spécification :

- d'une **métrique** qui mesure la similarité entre les deux images.
  On pendra ici la métrique `MeanSquaresImageToImageMetric` qui calcule l'erreur quadratique moyenne entre les images à recaler.
- d'une **transformation** qui contient les paramètres de recalage à estimer.
  On prendra ici la transformation `TranslationTransform` qui ne contient que les deux paramètres de translation (selon x et y)
- 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.TranslationTransform[itk.D, 2].New() # Instance de
↪  la classe de transformation choisie
initialParameters = initialTransform.GetParameters() # Récupération des
↪  paramètres de la transformation
initialParameters[0] = 0.0 # Translation selon x
initialParameters[1] = 0.0 # Translation selon y
```

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

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, nous appliquerons un ré-échantillonnage de l'image à recaler avant de l'écriture sur le disque :

```
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')
```

2.4 Travail à faire

1. En s'inspirant des extraits de codes ci-dessus, écrire un code permettant d'effectuer le recalage entre les deux images fournies

2. Enregistrer l'image recalée et comparer-la à l'image de référence pour vérifier que le recalage a bien fonctionné

3. Afficher la valeur de la métrique à convergence, le nombre d'itérations nécessaire, ainsi que la valeur des paramètres de la transformation.

4. Écrire sur le disque l'image de différence entre l'image recalée et l'image de référence.
   L'utilisation de `RescaleIntensityImageFilter` est requise.

3 Recalage par transformation rigide

Nous cherchons à effectuer le recalage de l'image *BrainProtonDensitySliceR10X13Y17.png* (moving) sur l'image *BrainProtonDensitySliceBorder20.png* (fixed), en utilisant une transformation rigide.
La transformation rigide utilisée sera `CenteredRigid2DTransform`.

1. Donner les paramètres de la transformation rigide

2. Initialiser le centre de rotation au centre de l'image

3. En vous inspirant de l'exercice précédent, réaliser le recalage entre les deux images

4 Recalage par information mutuelle

Nous cherchons à effectuer le recalage de l'image *BrainProtonDensitySliceR10X13Y17.png* (moving) sur l'image de référence *BrainT1SliceBorder20.png* (fixed), en utilisant une transformation affine.
La transformation rigide utilisée sera `itkSimilarity2DTransform`.
La métrique utilisée sera l'information mutuelle de Mattes :
`itkMattesMutualInformationImageToImageMetric`

1. Donner les paramètres de la transformation affine

2. Initialiser le centre de rotation au centre de l'image

3. Décrire le principe du calcul de l'information mutuelle

4. En vous inspirant des exercices précédents, réaliser le recalage entre les deux images

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

5 Panorama

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