TP Recalage ETI5
Module Image
15/11/2019

julien.jomier@kitware.com

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). Afin de configurer les exercices nous utiliserons CMake (http://www.cmake.org). Il est conseillé de se référer à la documentation d'ITK (compatible avec la version installée sur les machines) :

http://www.itk.org/Doxygen318/html/classes.html

Exercice 1 : ITK Hello World

Le but de cet exercice est de compiler une application en ITK qui crée une image et marque « Hello World ». Cette application permet juste de vérifier qu'ITK est bien compilé et fonctionne correctement.

1) Créer un fichier CMakeLists.txt

```
cmake_minimum_required(VERSION 2.6)
project(HelloWorld)

# Find ITK.
find_package(ITK)
if(ITK_FOUND)
  include(${ITK_USE_FILE})
else(ITK_FOUND)
  message(FATAL_ERROR
          "Cannot build without ITK.  Please set ITK_DIR.")
endif(ITK_FOUND)

add_executable(HelloWorld HelloWorld.cxx )
target_link_libraries(HelloWorld ITKCommon)
```

2) Créer un fichier HelloWorld.cxx

```
#include "itkImage.h"
#include <iostream>

int main()
{
  // Creer une image 3D avec un type de pixel unsigned short
  typedef itk::Image< unsigned short, 3 > ImageType;
  ImageType::Pointer image = ImageType::New();
```

```
  std::cout << "ITK Hello World !" << std::endl;
  return 0;
}
```

3) Lancer CMake sur le répertoire source
4) Compiler l'application et la lancer

Exercice 2 : Lecture d'une image

Le but de cet exercice est de lire l'image « Lena.jpg » et d'afficher la valeur du pixel (10,10) sur la console.

1) Modifier le code de l'exercice précédent afin de lire une image 2D de type « unsigned char » (pourquoi ce type de pixel?)
2) En s'inspirant du code suivant, changer le code afin de lire l'image Lena.jpg et vérifier que le code fonctionne correctement.

```
typedef itk::ImageFileReader< InputImageType >  ReaderType;
ReaderType::Pointer reader = ReaderType::New();
reader->SetFileName( "monimage.tiff" );
reader->Update();
InputImageType::Pointer image = reader->GetOutput();
```

Attention: vous devrez ajouter le header (.h) correspondant
Attention2: la bibliothèque ITKIO devra être ajoutée au linker

3) En s'inspirant du code suivant essayer de trouver la valeur du pixel (10,10)

```
InputImageType::IndexType index;
 index[0]=1;
 index[1]=1;
image->GetPixel(index);
```

4) *Question optionnelle : écrire le code qui permet de trouver la valeur du pixel (10,10) en se servant d'un iterateur. Quelle est la différence entre un itkImageRegionIterator et itkImageRegionConstIterator ?*

Exercice 3 : Lecture et écriture d'une image

Modifier le code de l'exercice précèdent afin d'écrire l'image qui vient d'être lu sur le disque dur sous le nom « imagesortie.jpg » et « imagesortie.bmp ». Utiliser la classe : itkImageFileWriter. (http://www.itk.org/Doxygen318/html/classitk_1_1ImageFileWriter.html)

Exercice 4 : Filtrage d'image

En utilisant le filtre itkRecursiveGaussianImageFilter, écrire un programme qui lit « lena.jpg » et applique un filtre Gaussien sur l'image. L'utilisateur doit pouvoir choisir les paramètres du filtre (quels sont-ils ?). Essayer avec d'autres images de votre choix.

Optionnel : Changer le filtre par un filter itkAbsoluteValueDifferenceImageFilter afin de calculer la différence entre deux images.

Exercice 5 : Recalage par Translation

En s'inspirant des bouts de code suivants, écrire un programme qui effectue le recalage de l'image **BrainProtonDensitySliceShifted13x17y.png** (moving) avec **BrainProtonDensitySliceBorder20.png** (fixed) en utilisant :

- Transformation : itkTranslationTransform
- Métrique : itkMeanSquaresImageToImageMetric
- Interpolateur : itkLinearInterpolateImageFunction
- Optimiseur: itkRegularStepGradientDescentOptimizer

La classe qui lie tous ces éléments est itkImageRegistrationMethod :

```
typedef itk::ImageRegistrationMethod<
                                    FixedImageType,
                                    MovingImageType    >  RegistrationType;


registration->SetOptimizer(     optimizer     );
registration->SetTransform(     transform     );
registration->SetInterpolator(  interpolator  );
registration->SetMetric( metric  );
registration->SetFixedImage( fixedImageReader->GetOutput() );
registration->SetMovingImage( movingImage );

registration->SetFixedImageRegion(
                           fixedImageReader->GetOutput()->GetBufferedRegion() );

typedef RegistrationType::ParametersType ParametersType;
ParametersType initialParameters( transform->GetNumberOfParameters() );

initialParameters[0] = 0.0;  // Initial offset in mm along X
initialParameters[1] = 0.0;  // Initial offset in mm along Y

registration->SetInitialTransformParameters( initialParameters );

try
  {
  registration->StartRegistration();
  }
catch( itk::ExceptionObject & err )
  {
  std::cout << "ExceptionObject caught !" << std::endl;
  std::cout << err << std::endl;
  return -1;
  }
```

Et pour afficher le résultat du recalage on utilisera la classe : itkResampleImageFilter :

```
ParametersType finalParameters = registration->GetLastTransformParameters();
TransformType::Pointer finalTransform = TransformType::New();
finalTransform->SetParameters( finalParameters );
```

```
ResampleFilterType::Pointer resample = ResampleFilterType::New();
resample->SetTransform( finalTransform );
resample->SetInput( movingImage );
resample->SetSize( fixedImage->GetLargestPossibleRegion().GetSize() );
resample->SetOutputOrigin(  fixedImage->GetOrigin() );
resample->SetOutputSpacing( fixedImage->GetSpacing() );
resample->SetDefaultPixelValue( 0 );
```

Optionnel : afficher le nombre d'itérations utiliser pour le recalage, la valeur des paramètres finaux et la valeur de la métrique.

Attention: vous devrez ajouter la bibliothèque ITKNumerics

Exercice 6 : Recalage par transformation rigide

A partir de l'exercice précédent, écrire un programme qui effectue le recalage de l'image **BrainProtonDensitySliceR10X13Y17.png** (moving) avec **BrainProtonDensitySliceBorder20.png** (fixed) en utilisant une transformation rigide (les transformations sont dans le dossier Insight/Code/Common).

Optionnel : Essayez avec différents types d'optimiseur.

Exercice 7 : Recalage par information mutuelle

A partir de l'exercice précédent, écrire un programme qui effectue le recalage de l'image **BrainProtonDensitySliceR10X13Y17.png** (moving) avec **BrainT1SliceBorder20.png** (fixed) en utilisant une transformation itkSimilarity2DTransform et la métrique d'information mutuelle par Mattes (itkMattesMutualInformationImageToImageMetric).

La transformation itkSimilarity2D a deux paramètres fixes qui définissent le centre de rotation :

```
ParametersType fixedParameters(2);
fixedParameters[0]= 120; // Centre en X
fixedParameters[1]= 120; // Centre en Y
transform->SetFixedParameters(fixedParameters);
```

Les paramètres sont optimisés de manière constante. Pensez à utiliser la fonction SetScale() pour que chaque paramètre ne soit pas optimisé de la même manière.

```
optimizer->SetScales(scales);
```

Exercice 8 : Panorama

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

Optionnel : Essayez avec deux images à vous !