

YIXUE TUXIANG FENGE YU PEIZHUN
ITK SHIXIAN FENCE

医学图像分割与配准

(② ITK实现分册)

周振环 王安明 王京阳 赵 明 著



电子科技大学出版社

医学图像分割与配准

(②ITK 实现分册)

周振环 王安明 王京阳 赵 明 著

医学图像分割与配准

(②ITK 实现分册)

著 周 明 王 京 阳 安 明 振 环

出 版 社：电子工业出版社
地 址：北京市西城区德胜门内大街5号
电 话：(010) 68289630
网 址：www.eel.com.cn
邮 政：100088
发 行：人民邮电出版社
印 刷：北京人民印刷厂
成 品 尺 寸：185mm×260mm
印 张：23.25
字 数：530千字
定 价：98.00元(含邮费)
书 号：ISBN 978-7-81114-271-7
印 次：2007年6月第1次印刷
发 行 次：2007年6月第1版

图 像 分 割 与 配 准

电子科技大学出版社

- ◆ 本书可作为医学图像处理、医学图像分割与配准课程的教学用书。
- ◆ 本书可作为从事医学图像处理、医学图像分割与配准工作的工程技术人员参考。
- ◆ 本书可作为从事医学图像处理、医学图像分割与配准工作的工程技术人员参考。

图书在版编目 (CIP) 数据

医学图像分割与配准 / 周振环等著. —成都: 电子科技大学出版社, 2007.6

ISBN 978-7-81114-571-7

I. 医… II. 周… III. 医学图像—图像数字化处理
IV. R445-39

中国版本图书馆 CIP 数据核字 (2007) 第 087411 号

医学图像分割与配准

(②ITK 实现分册)

周振环 王安明 王京阳 赵 明 著

出 版: 电子科技大学出版社 (成都市一环路东一段 159 号电子信息产业大厦 邮编: 610051)

策划编辑: 朱 丹

责任编辑: 张 鹏

主 页: www.uestcp.com.cn

电子邮件: uestcp@uestcp.com.cn

发 行: 新华书店经销

印 刷: 成都金龙印务有限责任公司

成品尺寸: 185mm×260mm 印张 33.5 字数 820 千字

版 次: 2007 年 6 月第一版

印 次: 2007 年 6 月第一次印刷

书 号: ISBN 978-7-81114-571-7

定 价: 98.00 元 (共两册)

■ 版权所有 侵权必究 ■

- ◆ 邮购本书请与本社发行部联系。电话: (028) 83202323, 83256027
- ◆ 本书如有缺页、破损、装订错误, 请寄回印刷厂调换。
- ◆ 课件下载在我社主页“下载专区”。

前 言

在 1999 年, 由美国国家卫生院 (NIH) 下属的国立医学图书馆 (NLM) 发起了一个投标活动, 出资赞助开发一个开放源码的分割与配准的算法研究平台。ITK 的 NIH/NLM 的工程负责人 Terry Yoo 博士带领 6 家单位合作开发, 这 6 家单位包括 GE Corporate R&D、Kitware、Inc. 和 MathSoft (现在公司改名为 Insightful) 三个商业公司和 University of North Carolina (UNC)、University of Tennessee (UT) (Ross Whitaker 随后迁往 University of Utah) 和 University of Pennsylvania (UPenn) 三所大学。

2002 年 ITK 官方首次发行 ITK 版本。这本书适用于 ITK2.4 更新版本。ITK 是一个开放源码、面向对象的软件系统, 提供一个医学图像处理、图像分割与配准的算法平台。虽然 ITK 结构庞大复杂, 但是一旦你了解它的面向对象和执行基本方法, 就可以灵活应用。这本软件指南的目的正是帮你了解这些方法以及这个平台中的主要算法和数据表达。书中已经提供了一些实例的使用资料, 你在阅读本书时便可以编译运行。

鉴于 ITK 是一个庞大的系统, 因此本书不可能完全介绍所有的 ITK 对象和方法。本书将尽最大能力指导你了解重要的系统概念, 并尽快尽好地指导你学习。ITK 是一个开放源码的软件系统, 这就意味着 ITK 用户和开发团体可以方便地对软件进行软件的开发和改进。

这本软件指南分为两部分, 每部分又包括几个章节。第一部分是 ITK 的基本情况介绍。第一章和接下来的两章介绍如何在你的计算机上安装 ITK, 包括安装预编译库和运行以及从源代码编译软件。第一部分同样也介绍了一些基本的系统概念, 如: 系统结构概述、如何使用 C++、Tcl 和 Python 编程语言建立应用程序。第二部分从用户角度来介绍软件, 提供了大量实例描述系统的主要特征。

ITK 用户可以明显地分为两类。第一类人是使用 C++ 创建新类的开发者, 另一类人是用已有的 C++ 类进行应用的使用者。类开发者必须非常精通 C++。如果他们要对 ITK 进行扩展和改进, 就必须非常熟悉 ITK 的内部结构和设计。作为 ITK 的使用者, 你必须了解 ITK 类和外部界面接口以及它们之间的关系。

学会使用 ITK 的关键就是熟悉各个对象的调色板和它们的方式。如果你是一位 ITK 的初学者, 那么就从安装软件开始学起吧。如果你是一位类开发者, 就需要安装源代码并编译。用户仅需预编译和执行程序。我们推荐你通过学习实例来了解系统。类开发者可学习源代码。先学习第三章提供的系统主要概念的综述, 然后回顾第二部分的实例。你也可以编译和运行这些实例。这些实例的源代码也可以在目录 Insight/Examples 中找到 (参见文件 Insight/Examples/README.txt 中包含的各种子目录里对这些实例的介绍)。在目录 Insight/Testing/Code 中的源代码分布里也可以找到许多测试, 这些测试大部分是测试代码。然而它们非常有助于了解 ITK 中类的使用方式, 尤其是它们尽可能地表达出了类的功能。

ITK 可以免费从以下网站下载 <http://www.itk.org/HTML/Download.php>。你可以得到一个稳定的版本或通过 CVS 得到比较新的版本。发行的版本比较稳定可靠但可能缺少研发平坦最新的特性功能。CVS 版本将含有最新的功能但有一定的不稳定性。

首先阅读 GettingStarted.txt 了解下载和安装进程。然后选择一个适合你系统的版本。有 .zip 和 .tgz 两个类型的文件供选择。第一种更适合于 MS-Windows 系统, 而第二种是 UNIX

系统的最佳版本。一旦你解压缩文件包，就将在你的电脑上生成一个称为 insight 的目录，你就可以按照书中描述的那样开始安装配置过程。

强烈建议你加入用户 mailing list。这是获得指导和使用帮助的主要来源。你可以在以下网站订阅拥护列表：<http://www.itk.org/HTML/MailingLists.htm>。用户 mailing list 同样也是表达你自己关于研发平台的观点和开发者了解有用的、期望的甚至是不必要的特征的最主要的机制。利用反馈的信息可以使 ITK 的开发者创立一个开放源码的 ITK 社区。

为了开始你的 ITK 之旅，首先你需要了解 ITK 的软件和目录结构。即使你安装了预编译库，这也将对你通过基本编码寻找实例、程序编码及文件有很大的帮助。

ITK 分为几个不同的模块或 CVS 校验区。官方或光盘版本有三大主要模块：Insight, InsightDocuments 和 InsightApplications 模块。Insight 模块包括有源代码、实例及应用；InsightDocuments 模块中包括有文件、指南和 ITK 的设计行销的有关材料；InsightApplications 模块中包含有 ITK（与其他如 VTK、Qt 和 FLTK 系统）的综合复杂应用。通常只需工作在 Insight 模块，开发者、培训课程者和许多具体细节的设计文件除外。只有当 Insight 模块的程序可以正常运行时才能下载和编译 InsightApplications 模块。

Insight 模块包含以下子目录：

- Insight/Auxiliary—工具包到 ITK 的界面代码。
- Insight/Code—软件的核心。主要源代码的位置。
- Insight/Documentation—用户开始 ITK 获得文献的简洁的子集。
- Insight/Examples—一系列样例、这本指南使用的和阐述重要 ITK 概念的文献样例。
- Insight/Testing—用来测试 ITK 的大量小程序。
- Insight/Code/Common—核心类、大量的定义、声明和 ITK 中其他重要的软件结构。
- Insight/Code/Numerics—数学库和支持类。
- Insight/Code/BasicFilters—基本的图像处理滤波器。
- Insight/Code/IO—支持读取和写数据的类。
- Insight/Code/Algorithms—大部分分割和配准算法的位置。
- Insight/Code/SpatialObject—使用空间关系表达和组织数据的类。
- Insight/Code/Patented—这里提供申请专利的算法，使用这些算法进行应用需要一个专利号。

- Insight/Code/Local—开发者使用的空目录，用户实验的新代码。

InsightDocuments 模块包含以下子目录：

- InsightDocuments/CourseWare—与 ITK 教学相关的材料。
- InsightDocuments/Developer—覆盖 ITK 设计和创建的历史文献，包括进程报告和设计文献。
- InsightDocuments/Web—<http://www.itk.org> 上找到的网站工具包的 HTML 源和其他材料。

InsightApplications 模块包含有大量相关的复杂的 ITK 使用实例。网页 <http://www.itk.org/HTML/Applications.htm> 上有相关的描述。

ITK 支持可视化人体工程（VHP）及它的相关数据。这些数据可以在国立医学图书馆的网站 http://www.nlm.nih.gov/research/visible/visible_human.html 上找到。

基金项目：2005 年粤港关键领域重点突破项目（2005A11304003）、深圳市科技计划项目（05KJcd004）联合资助。

目 录

第 2 部分

第八章 配准.....	3
8.1 配准框架.....	3
8.2 “Hello World” 配准	4
8.3 配准框架的特征.....	10
8.3.1 变换映射的方向.....	10
8.3.2 物理空间中的配准.....	11
8.4 监控配准.....	12
8.5 多形态配准.....	15
8.5.1 Viola-Wells 交互信息	15
8.5.2 粗糙的交互信息.....	20
8.5.3 绘制联合直方图.....	23
8.6 居中变换.....	26
8.6.1 二维刚性配准.....	27
8.6.2 采用图像力矩初始化.....	32
8.6.3 2D 相似变换.....	36
8.6.4 3D 中的刚性配准.....	38
8.6.5 中心仿射变换.....	42
8.7 多分辨率配准.....	45
8.7.1 主要原则.....	46
8.7.2 参数调节.....	51
8.8 变换.....	54
8.8.1 几何表示法.....	54
8.8.2 变换一般特征.....	56
8.8.3 一致变换.....	57
8.8.4 平移变换.....	57
8.8.5 比例转换.....	58
8.8.6 比例对数变换.....	59
8.8.7 欧拉 2D 变换.....	59
8.8.8 居中刚性 2D 变换.....	60
8.8.9 2D 相似度变换.....	60
8.8.10 四元数刚性变换.....	61

8.8.11	Versor 变换	61
8.8.12	Versor 刚体 3D 变换	62
8.8.13	欧拉 3D 变换	62
8.8.14	3D 相似变换	63
8.8.15	刚性 3D 透视变换	63
8.8.16	仿射变换	64
8.8.17	B 样条可变形变换	65
8.8.18	Kernel 变换	65
8.9	内插器	66
8.9.1	最近点差值	67
8.9.2	线性差值	67
8.9.3	B 样条插值	67
8.9.4	窗口化 Sinc 内插	68
8.10	尺度	69
8.10.1	均方 Metric	71
8.10.2	正则化相关尺度	73
8.10.3	倒数均方差分	73
8.10.4	互信息度量	74
8.10.5	Kullback-Leibler 距离 metric	76
8.10.6	规格化交互信息 Metric	76
8.10.7	直方图均方	76
8.10.8	相关系数直方图	76
8.10.9	基匹配度量	77
8.10.10	Kappa 统计 Metric	77
8.10.11	梯度微分 Metric	77
8.11	优化器	77
8.11.1	基度量的配准	79
8.11.2	1+1 进化优化配准	81
8.11.3	空间对象的模糊构建配准	82
8.11.4	结合预先知识的刚性配准	84
8.12	图像“金字塔”算法	85
8.13	可变形配准	86
8.14	虚拟变形配准	104
8.15	可视化变形域	110
8.15.1	可视化 2D 变形域	110
8.15.2	可视化 3D 变形域	112
8.16	基于配准的模型	114
8.17	点集配准	124

第九章 图像分割.....	136
9.1 区域生长.....	136
9.1.1 连接门限.....	136
9.1.2 OTSU 分割.....	139
9.1.3 邻域连接.....	141
9.1.4 置信连接.....	143
9.1.5 孤立连接.....	147
9.1.6 向量图像中的置信连接.....	149
9.2 基于分水岭的分割.....	151
9.2.1 综述.....	151
9.2.2 使用 ITK 分水岭滤波器.....	153
9.3 水平集分割.....	156
9.3.1 快速步进分割.....	158
9.3.2 形状检测分割.....	164
9.3.3 测量主动轮廓分割.....	171
9.3.4 阈值水平集分割.....	175
9.3.5 Canny 边缘水平集分割.....	177
9.3.6 拉普拉斯水平集分割.....	181
9.3.7 形状引导测量主动轮廓分割.....	183
9.4 混合法.....	192
9.4.1 绪论.....	192
9.4.2 模糊连接度和置信连接度.....	192
9.4.3 模糊连接度和 Voronoi 分类.....	194
9.4.4 可形变模型和 Gibbs 先验.....	199
9.5 特征提取.....	205
第十章 统计.....	213
10.1 数据箱.....	213
10.1.1 样本接口.....	213
10.1.2 样本改编器.....	215
10.1.3 直方图.....	219
10.1.4 子样本.....	222
10.1.5 成员样本.....	224
10.1.6 成员样本生成器.....	226
10.1.7 K-d 目录.....	229
10.2 算法和函数.....	233
10.2.1 样本统计.....	233
10.2.2 样本生成.....	237

10.2.3	样本分类.....	244
10.2.4	概率分布函数.....	247
10.2.5	间距量度.....	248
10.2.6	判断规则.....	249
10.2.7	随机变量的生成.....	251
10.3	应用于图像的统计学.....	252
10.3.1	图像直方图.....	252
10.3.2	图像信息理论.....	259
10.4	分类.....	265
10.4.1	基于 K 平均值堆积的 k-d 目录.....	266
10.4.2	K 平均值分类.....	271
10.4.3	贝叶斯插件分类器.....	273
10.4.4	最大期望混合模型估计.....	278
10.4.5	用 Markov 随机域进行分类.....	281

第 2 部分

医学图像分割与配准——ITK 实现

- 第 8 章 配准
- 第 9 章 图像分割
- 第 10 章 统计

第八章 配 准

本章主要是介绍 ITK 在图像配准方面的编程性能。图像配准是将一幅图像上的点映射到另一幅图像上同源点的空间转换过程。这种概念可以通过图 8-1 所示进行描述。在 ITK 中，配准就是在很容易被交换的限定的框架内进行编程。这种弹性意味着可以结合多种配准方法，以便允许使用者可以挑选并且选择对他们最适合的应用工具。

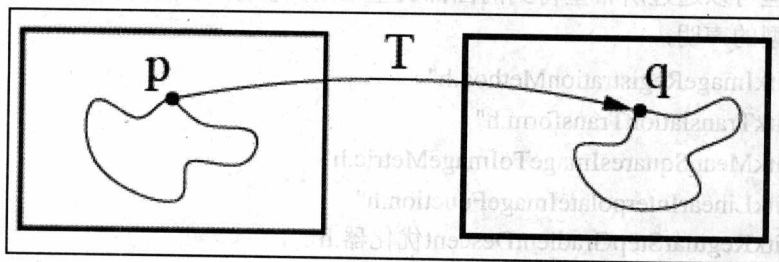


图 8-1 图像配准是同一目标的两幅图像在空间位置上的对准

8.1 配准框架

配准框架和它们之间的相互关系在图 8-2 所示中进行了描述。对于配准过程来说，基本的输入数据是两幅图：一幅被定义为参考图 $f(X)$ ，另一幅被定意为待配准图 $m(X)$ 。 X 表示在 N 维空间中的一个位置。配准被看成是一个找到把待配准图像映射到参考图像队列的最优化问题。

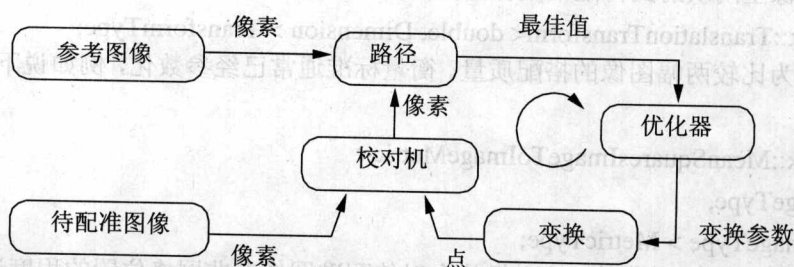


图 8-2 配准框架的基本成员：两个输入图像、一个变换、一个路径选择、一个校对机和一个优化器

传递函数 $T(X)$ 表示从参考图像上的点到待配准图像上的点的空间映射关系。校对机被用来评估待配准图像在非网格位置的程度。成员路径选择 $S(f, m \times T)$ 提供了一种参考图像被待配准图像配准的程度。这种尺度形成了数量上的标准，这种标准可以被优化器通过寻找被传递的参数定义的空间去达到最优化。

各种 ITK 配准组成将在以后部分进行描述，首先，我们从一些简单的配准例子开始。

8.2 “Hello World” 配准

这部分的源码可以在文件 Examples/Registration/ImageRegistration1.cxx 中。

这个例子表明了图像配准框架在 Insight 方面的应用，它对于 ITK 配准应该叫作 “Hello World”，这就意味着不用问 “为什么”。相反，这个例子可以用来说明在解决图像配准问题中典型的要素。

一个配准方法要求下面几条：两个输入的图像、转换函数、度量、校对机和优化器。这些要素中的一些可以通过所希望得到的图像类型来定义参数。下面的头文件提供了用在这些构成中普通类型的声明：

```
#include "itkImageRegistrationMethod.h"
#include "itkTranslationTransform.h"
#include "itkMeanSquaresImageToImageMetric.h"
#include "itkLinearInterpolateImageFunction.h"
#include "itkRegularStepGradientDescent优化器.h"
#include "itkImage.h"
```

在配准方法中的成分的每种类型都应该首先被实例化。为了达到这个目的，我们首先要选择图像的维数和图像像素的类型：

```
const unsigned int Dimension = 2;
```

```
typedef float PixelType;
```

输入数据的类型通过下面几行表达：

```
typedef itk::Image< PixelType, Dimension > FixedImageType;
```

```
typedef itk::Image< PixelType, Dimension > MovingImageType;
```

把参考图像空间映射到待配准图像空间的转换如下：

```
typedef itk::TranslationTransform< double, Dimension > TransformType;
```

衡量标准为比较两幅图像的匹配质量。衡量标准通常已经参数化，例如说下面对图像类型的声明：

```
typedef itk::MeanSquaresImageToImageMetric<
FixedImageType,
```

```
MovingImageType > MetricType;
```

最后，选择校对机的类型，校对机就会对待配准图像在非网格位置的程度进行评估：

```
typedef itk:: LinearInterpolateImageFunction<
```

```
MovingImageType,
```

```
double > InterpolatorType;
```

配准方法的类型是通过参考和待配准图像的类型来表示的。到目前为止，这类方法在连接我们所有已经描述的成分时是可靠的。

```
typedef itk::ImageRegistrationMethod<
```

```
FixedImageType,
```

```
MovingImageType > RegistrationType;
```

每一个配准要素都是通过它的New()创建的, 并且通过各自的itk::SmartPointer赋值。

```
MetricType::Pointer metric = MetricType::New();
```

```
TransformType::Pointer transform = TransformType::New();
```

```
优化器Type::Pointer 优化器 = 优化器Type::New();
```

```
InterpolatorType::Pointer interpolator = InterpolatorType::New();
```

```
RegistrationType::Pointer registration = RegistrationType::New();
```

每一个要素被连接到配准方法的程序中:

```
registration->SetMetric( metric );
```

```
registration->SetOptimizer( optimizer);
```

```
registration->SetTransform( transform );
```

```
registration->SetInterpolator( interpolator );
```

在这个例子里, 参考和待配准图像从文件里读取。这就需要itk::ImageRegistrationMethod从readers的输出中获得它的输入数据。

```
registration->SetFixedImage( fixedImageReader->GetOutput() );
```

```
registration->SetMovingImage( movingImageReader->GetOutput() );
```

如果仅仅考虑一些输入的参考图像的特别区域能够达到要求, 配准可能要受到一定的限制。这些区域通过SetFixedImageRegion()方法定义。你可以用这些模型减少配准的计算时间, 或者避免不想要的对象出现在图像中。这样的区域通过参考图像的BufferedRegion定义。注意, 在这个区域首先要调用它的Update()方法:

```
fixedImageReader->Update();
```

```
registration->SetFixedImageRegion(
```

```
fixedImageReader->GetOutput()->GetBufferedRegion());
```

变换参数通过将它们在队列里传递来初始化, 这能用来调整开始时引入的变化。在这种情况下, 平移变换用于配准。用于变换的参数队列由沿着每一维的方向的平移值构成。设置参数值到零以便将变换初始成恒等变换。注意: 队列的结构要求作为一个argument来传递元素的数量。

```
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 );
```

这时, 准备执行配准方法。优化器用来驱动配准的执行。然而, ImageRegistrationMethod类协调整体以确保在传递给优化器之前一切都到位了。

通常要对优化器的参数进行微调。每一个优化器都有特别的参数, 参数要在执行的优化策略里进行解释。这个例子里的优化器一个梯度下降的变量, 可以使振幅不要太大。每一次迭代, 优化器就会沿着itk::ImageToImageMetric派生的方向产生一个振幅。振幅的初始长度由用户定义。每次派生方向随机变化, 优化器假设局部极值已经被传递并且已经将振幅减为一半。在振幅减半若干次之后, 优化器也许将会在参数空间的有限区域内活动。用户可以定义振幅减小到多少可以看成收敛。这就相当于对最后的变换定义了一个精度。

初始振幅的长度用 `SetMaximumStepLength()` 定义，收敛域的公差用 `SetMinimumStepLength()` 定义：

```
optimizer->SetMaximumStepLength( 4.00 );
```

```
optimizer->SetMinimumStepLength( 0.01 );
```

这种情况下优化器不会得到我们想要的公差，建立迭代的次数需要谨慎。最大数用 `SetNumberOfIterations()` 定义：

```
optimizer->SetNumberOfIterations( 200 );
```

通过调用 `Update()` 函数触发配准程序的执行。如果在初始化或执行配准时出现错误，会出现异常提醒。因此我们放 `Update()` 函数在一个 `try/catch` 模块里：

```
try
```

```
{
```

```
registration->Update( );
```

```
}
```

```
catch( itk::ExceptionObject & err )
```

```
{
```

```
std::cerr << "ExceptionObject caught !" << std::endl;
```

```
std::cerr << err << std::endl;
```

```
return -1;
```

```
}
```

在现实的程序中，你也许想尽量通过采取一些措施以便从错误中恢复过来。这里我们简单地打出一个消息，并终结程序的执行。

配准程序的结果是一系列定义空间变换的参数序列。最终结果由 `GetLastTransformParameters()` 获得：

```
ParametersType finalParameters = registration->GetLastTransformParameters();
```

在 `itk::TranslationTransform` 里，有一个直接的参数的解释。队列的每一个元素对应着沿着一个空间维度的平移：

```
const double TranslationAlongX = finalParameters[0];
```

```
const double TranslationAlongY = finalParameters[1];
```

优化器能够询问抵达收敛的迭代的实际次数。`GetCurrentIteration()` 会返回这个值。迭代的次数太多说明也许最大振幅长度太小了，这样不太合适，因为它导致太长的计算时间。

```
const unsigned int numberOfIterations = optimizer->GetCurrentIteration();
```

相应地，最后参数集合的图像量规值能用优化器的 `GetValue()` 得到：

```
const double bestValue = optimizer->GetValue();
```

让我们运行以下的例子：

- `BrainProtonDensitySliceBorder20.png`

- `BrainProtonDensitySliceShifted13x17y.png`

第二幅图像是第一幅图像平移(13, 17)毫米后的结果。两幅图像都有单位间距，如图8-3所示。配准用了18次迭代，变换参数的结果是：

Translation X = 12.9959

Translation Y = 17.0001

这些值与我们预想的结果搭配相比要比较好。

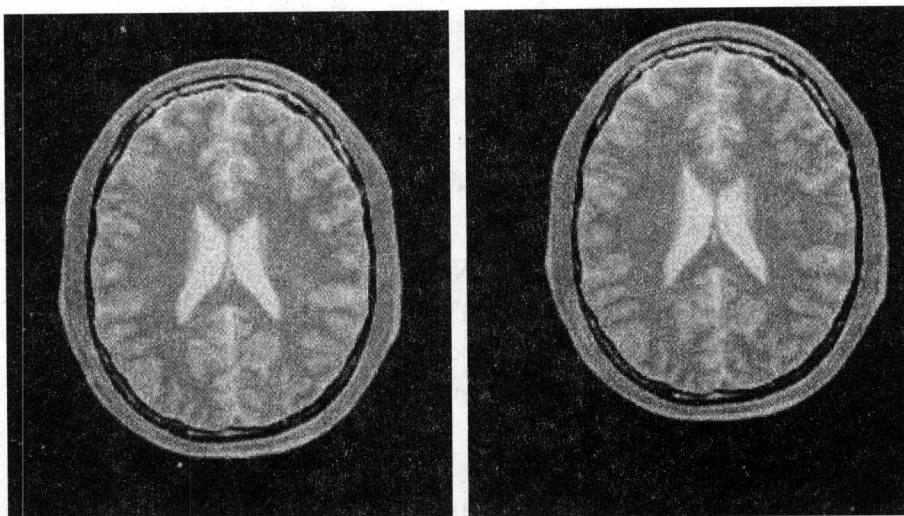


图 8-3 配准方法中作为输入的参考图像和待配准图像

普遍地，作为配准任务的最后振幅，用变换结果将待配准图像映射到参照图像空间。`itk::ResampleImageFilter`很容易做。可以参照6.9.4小节的滤波器的使用细节。首先，一个`ResampleImageFilter`类型以图像类型为例。用参照图像的类型作为输出类型很方便，因为被变换的待配准图像将会和参照图像进行比较。

```
typedef itk::ResampleImageFilter<
```

```
MovingImageType,
```

```
FixedImageType > ResampleFilterType;
```

一个重采样滤波器被创建待配准图像作为它的输入被连接。

```
ResampleFilterType::Pointer resampler = ResampleFilterType::New();
```

```
resampler->SetInput( movingImageReader->GetOutput() );
```

配准函数生成的变换也作为重采样滤波器的输入被传递。注意`GetOutput()`和`Get()`的使用。这里需要联合，因为配准函数作为一个滤波器，它的输出是一个`itk::DataObject`形式的变换。构建的细节你可以去读`itk::DataObjectDecorator`的文档。

```
resampler->SetTransform( registration->GetOutput()->Get() );
```

6.9.4小节中描述的是`ResampleImageFilter`要求指定额外的参数，特别是输出图像的间距、原点和大小。默认的像素值也设定了一个独特的灰度值以便加亮被映射出待配准图像外面的区域：

```
FixedImageType::Pointer fixedImage = fixedImageReader->GetOutput();
```

```
resampler->SetSize( fixedImage->GetLargestPossibleRegion().GetSize() );
```

```
resampler->SetOutputOrigin( fixedImage->GetOrigin() );
```

```
resampler->SetOutputSpacing( fixedImage->GetSpacing() );
```

```
resampler->SetDefaultPixelValue( 100 );
```

滤波器的输出被传递给一个在文件中存储图像的writer。`itk::CastImageFilter`转化重采样

的图像的像素类型到最终writer的像素类型:

```
typedef unsigned char OutputPixelType;
typedef itk::Image< OutputPixelType, Dimension > OutputImageType;
typedef itk::CastImageFilter<
FixedImageType,
OutputImageType > CastFilterType;
typedef itk::ImageFileWriter< OutputImageType > WriterType;
通过调用New()函数来创建滤波器:
WriterType::Pointer writer = WriterType::New();
CastFilterType::Pointer caster = CastFilterType::New();
滤波器被连接到一起, 并调用writer的Update()来触发管道的执行:
caster->SetInput( resampler->GetOutput() );
writer->SetInput( caster->GetOutput() );
writer->Update();
```

参照图像和被变换的待配准图像很容易用itk::SubtractImageFilter比较。pixel-wise滤波器计算两幅输入的同源像素的不同:

```
typedef itk::SubtractImageFilter<
FixedImageType,
FixedImageType,
FixedImageType > DifferenceFilterType;
DifferenceFilterType::Pointer difference = DifferenceFilterType::New();
difference->SetInput1( fixedImageReader->GetOutput() );
difference->SetInput2( resampler->GetOutput() );
```

注意: 作为比较图像的一种方法, 减法运算是合适的, 因为我们选择用浮点像素类型表示图像。如果图像的像素类型是无符号整型, 就要用一个不同的滤波器。

因为在两幅图像之间的不同也许符合较低的亮度值, 我们用itk::RescaleIntensityImageFilter重新调节亮度以使它们更加明显。如果我们保存不同的图像在一个只支持无符号整型像素的文件格式, 这种改变将会使负值可视化成为可能。我们也会把DefaultPixelValue减少到1, 以防两幅图像动态范围内的差异变小:

```
typedef itk::RescaleIntensityImageFilter<
FixedImageType,
OutputImageType > RescalerType;
RescalerType::Pointer intensityRescaler = RescalerType::New();
intensityRescaler->SetInput( difference->GetOutput() );
intensityRescaler->SetOutputMinimum( 0 );
intensityRescaler->SetOutputMaximum( 255 );
resampler->SetDefaultPixelValue( 1 );
它的输出被送到另一个复写器中:
WriterType::Pointer writer2 = WriterType::New();
```