Journal of Statistical Software

August 2018, Volume 86, Issue 8. *doi: 10.18637/jss.v086.i08*

Image Segmentation, Registration and Characterization in R with SimpleITK

Richard Beare
Monash University,
Murdoch Childrens
Research Institute

Bradley Lowekamp
National Institutes of Health,
Medical Science and
Computing

Ziv Yaniv
National Institutes of Health,
TAJ Technologies Inc.

Abstract

Many types of medical and scientific experiments acquire raw data in the form of images. Various forms of image processing and image analysis are used to transform the raw image data into quantitative measures that are the basis of subsequent statistical analysis.

In this article we describe the **SimpleITK** R package. **SimpleITK** is a simplified interface to the insight segmentation and registration toolkit (**ITK**). **ITK** is an open source C++ toolkit that has been actively developed over the past 18 years and is widely used by the medical image analysis community. **SimpleITK** provides packages for many interpreter environments, including R. Currently, it includes several hundred classes for image analysis including a wide range of image input and output, filtering operations, and higher level components for segmentation and registration. Using **SimpleITK**, development of complex combinations of image and statistical analysis procedures is feasible. This article includes several examples of computational image analysis tasks implemented using **SimpleITK**, including spherical marker localization, multi-modal image registration, segmentation evaluation, and cell image analysis.

Keywords: image processing, image segmentation, image registration, medical imaging, R.

1. Introduction

Images are an important source of quantitative information in many fields of research and many industrial, medical and scientific applications. The processing required to transform pixel or voxel data into quantitative information used in subsequent statistical analysis is as varied as the imaging instruments used to acquire image data, the scenes being imaged and the form of information required. Instruments in typical use include basic digital cameras in scenarios like security and wildlife monitoring, digital cameras associated with an ever increasing variety of microscopes, and medical imaging equipment such as magnetic resonance imaging (MRI), computed tomography (CT), ultrasound and X-ray. Other instruments, such as LIDAR, produce gridded data that may be useful to regard as an image in some scenarios. This small list of examples serves to illustrate the range of data types typically acquired, before considering the range of scenes being imaged. Typical digital cameras are capable of acquiring two-dimensional scenes, possibly with color channels and possibly with time series. MRI is capable of acquiring a range of three-dimensional (3D) time series and other, more specialized, forms including directional information in 3D. Microscopes are capable of acquiring time series of 3D images with multiple fluorescent channels, and such microscopes are being coupled to automated sample preparation systems to produce automated screening platforms capable of generating massive amounts of images from complex experiments.

The type of information extracted from an image may range from a detection (e.g., presence of an intruder or an animal) to a count (e.g., number of cells or brain lesions) to more complex characterizations of objects (e.g., measures of length/area/volume, spatial distributions, brightness) to tracks of moving objects in time series. There are numerous options, and the difficulty varies considerably with scene complexity and consistency.

Creating a new image analysis process typically requires experimentation with a variety of approaches, each of which will combine several computational steps or components. It is a considerable advantage to have on hand an image analysis framework with a large number of functional components capable of operating on multi-dimensional data when embarking on such an endeavor. The Insight Segmentation and Registration Toolkit (**ITK**; Johnson, McCormick, Ibáñez, and The Insight Software Consortium 2013) is such a framework, offering thousands of components for image input and output, image filtering, image segmentation and image registration. Originally designed for medical images, **ITK** has been successfully used in varied domains with data ranging from cell images to remote sensing images.

In this article we introduce the **SimpleITK** package (The Insight Software Consortium 2018) for R(R Core Team 2018), which allows development of image analysis pipelines using R and a simplified interface to **ITK**.

2. ITK, SimpleITK and the SimpleITK R package

2.1. ITK

ITK is a large, open source, C++ library which includes a wide variety of components for image analysis. The toolkit was originally developed for analysis of medical images, as part of the Visible Human Project at the National Library of Medicine, USA (Yoo *et al.* 2002). **ITK** employs a generic design enabling support for arbitrary image dimensions and pixel types. The toolkit was funded by the U.S. National Institutes of Health to provide a well engineered platform for development of new algorithms and allow testing of existing algorithms on new data, without the cost of redeveloping published methods. There are currently 1800+ classes available.

The toolkit is widely used both as a foundation for other toolkits and as a component in applications for analysis of medical images. Amongst others, these include toolkits for computer assisted surgery such as **MITK** (Wolf *et al.* 2005) and **IGSTK** (Enquobahrie *et al.* 2007), and toolkits for image registration such as **elastix** (Klein, Staring, Murphy, Viergever, and Pluim 2010), **ANTS** (Avants, Tustison, Song, Cook, Klein, and Gee 2011) and **BRAINSFit** (Johnson, Harris, and Williams 2007). Examples of medical image analysis applications developed using **ITK** include **ITK-SNAP** (Yushkevich *et al.* 2006), **3D Slicer** (Fedorov *et al.* 2012), and **medINRIA** (Toussaint, Souplet, and Fillard 2007). Outside of the medical domain it is used by the remote sensing toolbox, **ORFEO** (Inglada and Christophe 2009), and the cell image analysis tool **GoFigure2** (Mosaliganti, Gelas, and Megason 2013).

ITK has two features distinguishing it from most image analysis toolkits:

- Complex operations are accomplished by using filter pipelines.

  Filter pipelines are the architecture used in **ITK** to accomplish complex tasks, such as segmentation, via lazy evaluation. A pipeline of filters is constructed to implement a series of computational steps, however no computation is performed until a call is made to the `Update` method of the last filter. The request is propagated back along the pipeline and only the required processing is performed. Pipelines also facilitate processing of datasets that are too large for core memory via streaming.

- An image occupies a region in physical space and operations should take this into account.

  The center of every voxel in an **ITK** image is located at a specific point in space with respect to a coordinate system, the axes of which are not necessarily orthogonal. Distances between adjacent voxel centers are in physical units and are not necessarily isotropic.

ITK offers a high degree of customizability which, unfortunately, leads to a steep learning curve for developers and requires significant C++ experience.

2.2. SimpleITK

By making several judicious design decisions, **SimpleITK** is able to provide a simplified interface to **ITK** while maintaining most of the functionality available in the original toolkit. The key design decisions were based on the following observations obtained via surveys: most users analyze two-dimensional (2D), three-dimensional (3D), and possibly four-dimensional images (3D plus time); most users do not take advantage of the features of the pipeline approach; non-C++ developers prefer a procedural interface over an object oriented one.

As a consequence, **SimpleITK** was designed to support 2D and 3D images, with an optional configuration flag for 4D support. The pipeline architecture is hidden in favor of immediate execution, and both, an object oriented and a procedural interface, have been implemented. Note that while the image dimensions are restricted, voxels can be vectors of arbitrary length containing a multi-dimensional measure, such as color, fiber orientation or spectra.

Just like **ITK**, **SimpleITK** is implemented in C++. Unlike **ITK**, most of the code is generated automatically using a JavaScript Object Notation (JSON) template structure. A detailed description of the design and automated code generation process is given in Lowekamp, Chen, Ibáñez, and Blezek (2013). The design of **SimpleITK** also allows for easy automated wrapping of the C++ implementation for several interpreted languages including Python(Rossum *et al.* 2011), Perl (Christiansen, Foy, Orwant, and Wall 2012), C#, Lua (Ierusalimschy 2016), Ruby (Thomas, Fowler, and Hunt 2009), Tcl (Tcl Core Team 2017), and R, the focus of this article.

Automated generation of interface code is critical for long term viability of **SimpleITK** due to its dependence on **ITK**, which is very large and continually evolving.

2.3. Licensing

SimpleITK is distributed under the Apache 2.0 license. The full license is available at `http://www.simpleitk.org/SimpleITK/project/license.html`.

2.4. The SimpleITK R package

Multiple layers of automated code generation and dependencies on external tools make building **SimpleITK** a challenge and cause problems distributing packages via traditional mechanisms, such as at the Comprehensive R Archive Network (CRAN). Two approaches for Linux and OSX systems are described below.

Documentation

Documentation is automatically converted from the doxygen extracted C++ class documentation. It offers a non-standard starting point for the R developer. Most important details are to be found in the help for the class interface, rather than the help for the procedural interface. For example `?Cast` will display only the most basic information concerning usage (useful for determining argument names and order) while details of functionality are available in `?CastImageFilter`. This division is common across **SimpleITK** and is shared with the C++ documentation. The C++ documentation is structured to describe the classes and associated methods, and thus does not fit into the R function documentation design. The current approach maps class methods to function arguments. The **SimpleITK** C++ documentation (The Insight Software Consortium 2016b) is the canonical source of information concerning available methods and classes.

Beyond the API documentation described above the toolkit also maintains general, language agnostic, documentation on read-the-docs `https://simpleitk.readthedocs.io/`. This documentation covers installation, fundamental concepts specific to **SimpleITK**'s image and transformation elements, common API conventions, frequently asked questions and short example programs. Additional resources include a Jupyter notebook repository illustrating complete image-analysis workflows in Python and R (Yaniv, Lowekamp, Johnson, and Beare 2018, ; `https://github.com/InsightSoftwareConsortium/SimpleITK-Notebooks`) and a discourse discussion forum for users to post questions (`https://discourse.itk.org/`).

Installation using ***devtools***

The simplest way to install the **SimpleITK** package from source is using **devtools** (Wickham, Hester, and Chang 2018) to fetch a stub installer package from GitHub and build it. This procedure is dependent on CMake, make, Git, C++ compilers and **devtools**. The following will fetch, build and install the package.

```
R> devtools::install_github("SimpleITK/SimpleITKRInstaller")
```

Additional help and answers to common problems are provided at `http://github.com/SimpleITK/SimpleITKRInstaller`

Building from source

Greater control of the build process, as well as the ability to generate other wrapper packages and run testing procedures, is available with a source build.

Building of the R package involves two code generation steps. In the first step the C++ classes are generated using scripts written in the Lua programming language. These classes are described by a combination of template and JSON files describing each of the classes. Some additional classes are implemented directly in C++. Once all of the C++ source is created, **SWIG** is used to generate the R bindings for all classes.

The current build process requires CMake and Git to fetch and build dependencies. The additional components on which the build process depends include **SWIG**, Lua, and **ITK**. For a fully automated build process without any pre-installed components one selects the project's SuperBuild CmakeLists file without making any change to the CMake settings. The build process is documented at The Insight Software Consortium (2016a). In brief, the process is:

- Clone the source code:

  ```
  $ git clone http://itk.org/SimpleITK.git
  ```

- Create a build folder and run `cmake` on the SuperBuild subdirectory:

  ```
  $ mkdir SimpleITK-build
  $ cd SimpleITK-build
  $ cmake ../SimpleITK/SuperBuild
  $ make
  ```

The build process fetches and builds all dependencies. A successful build produces a binary R package in `SimpleITK-build/Wrapping/R` that can be distributed and installed.

Wrapper generation with ***SWIG***

The software interface generator, **SWIG** (Beazley *et al.* 1996), was originally designed to generate Python interfaces to C code, but now supports many interpreted environments, including R. This makes **SWIG** a natural choice as the interface generator for **SimpleITK**, which aims to support multiple interpreted programming languages. However **SWIG**'s popularity in the R world is probably lower than **Rcpp** (Eddelbuettel and François 2011), which provides an API layer for C++/R rather than an interface generator. The **SWIG** support for R has been extended to support **SimpleITK** development, specifically support for C++ `stl` vectors (automatic conversion to and from R vectors). These developments potentially make **SWIG** an interesting alternative to **Rcpp** for projects targeting multiple programming languages in addition to R.

Objects in **SimpleITK**, such as images and filters, are *external references* to C++ objects. **SWIG** generated binding code uses a combination of S4 classes and a reference class style structure to wrap these objects which are then managed via the R garbage collection system.

Multi-threaded components

There are a number of architectural features in **ITK** to make construction of multi-threaded filters simpler. As a result a substantial proportion of **SimpleITK** components are able to take advantage of multicore processors and multiple processor configurations. This includes all operations that operate on single pixels, such as image arithmetic, many kernel filters and parts of the registration framework. By default the number of threads a component uses is equal to the number of cores available. The number of threads used by a class can be queried and set as shown in the following code snippet.

```
R> reg <- ImageRegistrationMethod()
R> reg$GetNumberOfThreads()

[1] 8

R> reg$SetNumberOfThreads(2)
R> reg$GetNumberOfThreads()

[1] 2
```

3. SimpleITK concepts

3.1. Image files, image objects and meta-data

SimpleITK can read and write many image file formats, including traditional 2D formats such as jpeg, bmp, png and tiff, as well as medically oriented, 3D formats including Digital Imaging and Communications in Medicine (DICOM), analyze, nifti, nrrd, metaimage and others. The image format is automatically determined during reading and writing.

Images in **SimpleITK** are *external references*, that is pointers to C++ objects. They are created by loading data from files, calls to constructor methods or conversion from R arrays. Image meta-data, such as voxel type, voxel spacing, and image orientation may be queried and set using method calls, as follows:

```
R> img <- ReadImage("phantom.dcm")
R> img$GetSize()

[1] 512 512   1

R> img$GetWidth()

[1] 512

R> img$GetSpacing()

[1] 0.65625 0.65625 1.00000
```

This DICOM image is a single 512×512 slice, with voxel dimensions $0.65625 \times 0.65625 \times 1$mm. The presence of slice thickness information in the DICOM file leads to creation of a 3D image, even though there is only a single slice. Image formats, such as png, which typically do not have thickness information will be interpreted as 2D images.

The meta-data can also be modified, for example to change the spacing:

```
R> img$SetSpacing(c(2, 2, 2))
R> img$GetSpacing()

[1] 2 2 2
```

An image also contains a meta-data dictionary. In many cases this meta-data dictionary is empty, but for some images, such as those stored using the format specified by the DICOM standard, this meta-dictionary contains a considerable amount of information, including patient name, imaging modality, equipment manufacturer and many other data elements as shown below. **SimpleITK** allows us to query and set this dictionary as follows:

Load a DICOM image and retrieve all the meta-data dictionary keys:

```
R> img <- ReadImage("phantom.dcm")
R> head(img$GetMetaDataKeys(), 5)

[1] "0008|0000" "0008|0005" "0008|0008" "0008|0012" "0008|0013"
```

Query a specific metadata key – e.g., image modality – and modify it:

```
R> img$GetMetaData("0008|0060")

[1] "CT"

R> img$SetMetaData("0008|0060", "JSS Example")
R> img$GetMetaData("0008|0060")

[1] "JSS Example"
```

Other commonly used medical image formats also have meta-data dictionaries accessible in the same way. Nifti files, for example, have dictionary entries such as `descrip`, `qform_code_name` etc.

Image extent and physical coordinates

Relationships between image data and the patient are critical when working with medical images. It is usually important to know the precise size of a voxel, and sometimes exactly where it was located in the scanner. Such knowledge is clearly needed when measuring sizes of pathology, such as tumors, that are expected to change size over time. It is also critical in applications where images are combined, to ensure that only legal operations are performed – for example, performing masking with images of the same size but different orientations is likely to cause errors. **SimpleITK** provides image orientation and image voxel spacing information that enable these operations and ensure that illegal operations are detected.

The key components are image origin, voxel spacing and axis orientation. The following code snippet illustrates the creation of a 2D axis aligned image with non-zero origin and non-isotropic pixel spacing and demonstrates mapping between voxel and physical units. Looking at the returned indexes we see that the pixel extent in physical space indeed starts half a pixel from its center and that the indexes in **SimpleITK** are zero-based.

```
R> img <- Image(64, 64, "sitkFloat32")
R> img$SetOrigin(c(0.3, 0.6))
R> img$SetSpacing(c(0.4, 1.0))
R> p1 <- c(0.1, 0.1)
R> p1Indexes <- img$TransformPhysicalPointToIndex(p1)
R> cat("Point coordinates: [", p1, "] Point indexes: [", p1Indexes, "]\n")

Point coordinates: [ 0.1 0.1 ] Point indexes: [ 0 0 ]

R> p2 <- c(0.0, 0.1)
R> p2Indexes <- img$TransformPhysicalPointToIndex(p2)
R> cat("Point coordinates: [", p2, "] Point indexes: [", p2Indexes, "]\n")

Point coordinates: [ 0 0.1 ] Point indexes: [ -1 0 ]

R> p3 <- c(0.1, 0.0)
R> p3Indexes <- img$TransformPhysicalPointToIndex(p3)
R> cat("Point coordinates: [", p3, "] Point indexes: [", p3Indexes, "]\n")

Point coordinates: [ 0.1 0 ] Point indexes: [ 0 -1 ]
```

3.2. Display

Printing the image object using the `print` method will display the image meta-data and associated underlying internal data structures.

SimpleITK has no inherent image display capabilities. A convenience `Show` function is provided, displaying images by writing them to a temporary location in nifti format and then invoking an external viewer. By default this function will attempt to invoke the **Fiji/ImageJ** viewer. Invoking other viewers is facilitated by setting `SITK_SHOW_COMMAND` as environment variable to indicate a specific viewer. Finer grained control is possible, allowing one to specify a viewer for color images and for 3D images by setting the `SITK_SHOW_COLOR_COMMAND` and `SITK_SHOW_3D_COMMAND` environment variables.

3.3. Conversion to and from arrays

Images can be converted to native R arrays using `as.array` and arrays can be converted to images using `as.image`. Image meta-data can be copied from one image to another or set manually. Note that `as.array` includes a `drop` option to discard unit dimensions.

3.4. Indexing and slicing

Unlike R indexing, **SimpleITK** functions use zero-based indexing, so you will need to account for this when working with **SimpleITK** images. The exception to the rule is slicing. This is a native R operation and **SimpleITK** supports it using R's one-based indexing:

```
R> img <- Image(1, 1, "sitkUInt8")
R> rIndex <- 1
R> img[rIndex, rIndex]
```

```
[1] 0
```

```
R> img$SetPixel(c(rIndex, rIndex) - 1, 255)
R> img[rIndex, rIndex]
```

```
[1] 255
```

Slicing allows complex manipulation of images using standard syntax, returning new images. Cropping, flipping, etc. is easy to achieve using slicing operations:

- Extracting one corner:

  ```
  R> img <- Image(10, 10, "sitkUInt8")
  R> imgCorner <- img[1:5, 1:5]
  R> imgCorner$GetSize()
  ```

  ```
  [1] 5 5
  ```

- Simple reflection:

  ```
  R> imgPadLeft <- img[img$GetWidth():1, ]
  R> imgPadLeft$GetSize()
  ```

  ```
  [1] 10 10
  ```

- Accessing a single pixel:

  ```
  R> pix <- img[2, 3]
  R> pix
  ```

  ```
  [1] 0
  ```

- Illegal operation – remove central columns producing non-uniform spacing:

  ```
  R> ii <- img[-c(3:5), ]
  ```

  ```
  Error in img[-c(3:5), ]: X spacing is not uniform
  ```

Image slicing is designed to preserve image constraints, and is thus slightly less flexible than standard R array indexing. A slicing operation must produce images with uniform voxel spacing along each image dimension. The concept of empty arrays, produced by indexing operations like `ar[0, 1:2]`, does not translate directly to image objects, and an error is raised. A voxel is converted to a native R object when a slicing operation produces a single voxel result (rather than returning a single voxel image). Slicing of multi-component images is possible, with multi-component voxels being returned as R vectors. Array-based indexing is not yet supported.

3.5. Image operators

Operator overloading is used to simplify arithmetic and logical expressions with images. The operations are carried out by **SimpleITK** filters, offering control over pixel types, multi-threaded operation and meta-data consistency. For example:

```
R> sz <- 512
R> im1 <- Image(sz, sz, sz, "sitkFloat64") + 1
R> arr2 <- array(seq(1, sz^3, 1.0), c(sz, sz, sz))
R> im2 <- as.image(arr2)
R> im3 <- Image(sz, sz, sz, "sitkFloat64") + 2
R> mask <- ((im1 * im2) / im3) > 24
```

3.6. Methods for SimpleITK objects

There are some useful R language tricks that can be used to interrogate image and filter objects to identify available methods. For example:

```
R> getMethod("$", class(img))

Method Definition:
function (x, name)
{
    accessorFuns = list(Equal = Image_Equal, GetITKBase = Image_GetITKBase,
        GetPixelID = Image_GetPixelID,
        GetPixelIDValue = Image_GetPixelIDValue,
        GetDimension = Image_GetDimension,
        GetNumberOfComponentsPerPixel = Image_GetNumberOfComponentsPerPixel,
        GetNumberOfPixels = Image_GetNumberOfPixels,
        GetOrigin = Image_GetOrigin, SetOrigin = Image_SetOrigin,
        GetSpacing = Image_GetSpacing, SetSpacing = Image_SetSpacing,
        GetDirection = Image_GetDirection,
...
```

3.7. Error messages

The reference class style of method call, combined with the need for method overloading, leads in some cases to unhelpful error messages, such as when invoking a non-existent method:

```
R> img <- Image(10, 10, "sitkUInt8")
R> img$SetVoxel(c(5, 5), 255)

Error in callNextMethod() : node stack overflow
```

Or passing combinations of arguments that the method overloading code is unable to resolve:

```
R> th <- Threshold(img, "pixtype")

Error in f(...): could not find function "f"
```

In the first example we should be using `SetPixel` while in the latter case there is no function call version of `Threshold` with image and character arguments.

Finally, configuration of the external viewer often causes problems the first time one uses **SimpleITK**. The primary reason being that the default viewers, **ImageJ/Fiji**, are not installed and the environment variable specifying an alternative viewer was not set as described above.

4. SimpleITK computational components

SimpleITK reduces the learning curve compared to **ITK** by simplifying many of the programming aspects. Unfortunately, mastering all of the available functionality still requires an effort, primarily due to the size of the computational framework.

It is often difficult to identify components of interest due to differences in naming conventions between toolkits. **ITK** uses a module hierarchy for code that can help make the range of components more comprehensible. Not all of the classes are directly accessible from **SimpleITK**, but most are or will eventually be. Infrastructure modules have been left out for clarity. The key modules and their contents are described below.

4.1. Filtering

This is the largest module and contains numerous classes that modify voxel values in some way. It contains the following categories, in alphabetical order:

- AnisotropicSmoothing: Gradient and curvature anisotropic diffusion for scalar and vector images.
- AntiAlias: Reduce aliasing artifacts when displaying binary volume.
- BiasCorrection: The N4 algorithm for bias field inhomogeneity in MR images.
- BinaryMathematicalMorphology: Specialized mathematical operations for binary images.
- Colormap: Colormaps for creating overlays and display.
- Convolution: Convolution and correlation classes.
- CurvatureFlow: Diffusion-based denoising.
- Deconvolution: A range of deconvolution algorithms – RichardsonLucy, Tikhonov, etc.
- Denoising: Patch-based denoising.
- DiffusionTensorImage: Basic tools for diffusion tensor data – reconstruction, fractional anisotropy.
- DisplacementField: Tools for processing displacement images produced by registration.
- DistanceMap: Signed and unsigned distance transforms using various algorithms.
- FastMarching: Classes to compute geodesic distances on images using the fast marching algorithm.
- FFT: A range of fast Fourier transform classes.

- ImageCompare: Checkerboard visualization aids.
- ImageCompose: Tiling, joining.
- ImageFeature: Edge detection, Laplacians, Hessians, etc.
- ImageFusion: Overlays.
- ImageGradient: Separable, recursive gradient filters.
- ImageGrid: Padding, cropping, resampling etc.
- ImageIntensity: Arithmetic, logical operations, etc.
- ImageLabel: Manipulation of label images (outputs of connected component analysis).
- ImageNoise: Noise generators for algorithm tests.
- ImageSources: Filters creating various sorts of synthetic images (e.g., coordinate positions).
- ImageStatistics: Whole image statistics, label statistics, projections, overlap measures.
- LabelMap: Manipulation of run-length-encoded objects from connected component analysis.
- MathematicalMorphology: Erosions/dilations using various fast decompositions, tophat filters, geodesic reconstruction, regional extrema and attribute morphology.
- Smoothing: Separable, recursive smoothing filters.
- Thresholding: Threshold estimation using a range of histogram-based approaches.

4.2. Segmentation

Image segmentation filters produce output images in which voxel values indicate class membership. The relevant ITK modules are:

- Classifiers: Bayesian, K-means voxel classifiers.
- ConnectedComponents: Label objects in a binary image (aka particle analysis).
- DeformableMesh: Mesh-based segmentation in which mesh deformation is driven by image forces.
- KLMRegionGrowing: Koepfler, Lopez and Morel algorithm.
- LabelVoting: Various schemes for combining label images, including STAPLE, voting. Hole filling with voting.
- LevelSets: An extensive framework for segmentation using the level set methodology. Includes geodesic active contours, shape priors and others.

- MarkovRandomFieldsClassifiers: MRF voxel classification class.
- RegionGrowing: Various combined threshold and connectivity approaches to segmentation.
- SignedDistanceFunction: Distance functions for shape models.
- Voronoi: Color segmentation tools using Voronoi tessellation.
- Watersheds: Several algorithms for watershed transforms, including marker-based options.

4.3. Registration

Image registration is the process of estimating a spatial transformation that makes two images similar according to some measure. It is usually structured as an optimization problem. There are numerous choices available for similarity measures, optimizers, and transformation functions. For intensity-based registration, **SimpleITK** makes many of these available via a single framework class, with options for callback functions to track optimizer progress. The available choices include:

- Similarity metrics: Correlation, mutual information (Mattes and joint histogram), Demons, mean squares, and the ANTS neighborhood correlation.
- Transforms: Rigid, similarity, affine, B-spline, dense deformation field, etc.
- Optimizers: Conjugate gradient line search, gradient descent, exhaustive, LBFGSB, etc.
- Interpolators: Nearest neighbor, linear, B-spline, windowed sinc, etc.

Classes implementing transforms, optimizers and interpolators are in core modules, rather than the registration module, as they are used in other scenarios.

Additional algorithm specific implementations are also available. These include the implementation `LandmarkBasedTransformInitializer`, estimating the transformation which minimizes the distance between two point sets with known correspondences (transformation can be rigid, affine or B-spline); and several Demons-based intensity-based algorithms for estimating the dense deformation field between images, `Demons`, `DiffeomorphicDemons`, `SymmetricForcesDemons` and `FastSymmetricForcesDemons`.

5. Case studies

The addition of **SimpleITK** to the R development environment enables the rapid development of end-to-end image characterization and statistical analysis tools. This combination benefits from the large choice of image operators available in **SimpleITK** and the extensive range of existing R packages for feature extraction and statistical analysis. In addition, the use of **SimpleITK** in an interpreted environment offers a reduction in development cycle time by removing the compile stage from the usual change-compile-test cycle.

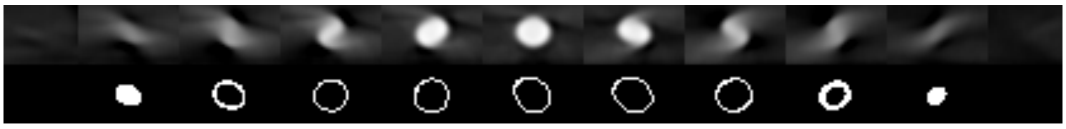


Figure 1: Cropped cone-beam CT volume of a metallic sphere (top row) and the result of performing 3D edge detection on the volumetric data (bottom row). Original image intensity values have been mapped to [0, 255] for display purposes. The filled circle edge images at both ends (left, right) of the sphere highlight the fact that the operation is indeed carried out in 3D. If performed on a slice by slice manner all edge images would result in empty circles.

In this section we will illustrate **SimpleITK** via several example case studies, with the aim of providing an introduction to both, the slightly unusual syntax of a **SWIG** interface and some of the extensive capabilities of **SimpleITK**. These examples illustrate only a small proportion of the available classes but provide an introduction to several components that are useful in many different scenarios.

5.1. Spherical marker localization

Alignment between a patient and volumetric images of that patient is critical in many forms of surgical/medical intervention. Spherical markers that are visible on the patient and in the volumetric images are frequently used to aid registration in computer assisted intervention. Before the markers can be used for alignment they need to be localized in the image coordinate system as described in Yaniv (2009). There are a variety of approaches for spherical marker localization. In our case we will perform edge detection using **SimpleITK**, and use R to obtain a least-squares estimate of the sphere's location and radius.

Our input image was acquired using a cone-beam CT system and contains a single sphere. The image is non-isotropic, a $40 \times 25 \times 11$ volume with a $0.44 \times 0.44 \times 0.89$mm spacing, and has a high dynamic range of intensity values, $[-32767, -25481]$. Figure 1 shows the volume containing the metallic sphere and the result of performing edge detection on it.

The localization code, below, employs edge detection with parameters selected to match the anisotropic image voxel spacing.

First, load the image.

```
R> library("SimpleITK")
R> sphere_image <- ReadImage("sphere.mha")
```

Second, perform 3D edge detection to produce a binary edge map using a Canny edge detector.

```
R> edges <- CannyEdgeDetection(Cast(sphere_image, "sitkFloat32"),
+    lowerThreshold = 0.0, upperThreshold = 200.0,
+    variance = c(5.0, 5.0, 10.0))
```

Third, convert the edge image to an array and determine the physical coordinates of each edge voxel using the built-in image method `TransformIndexToPhysicalPoint`. Note that indexes are adjusted to zero-based to match C++ standards.

```
R> edge_indexes <- which(as.array(edges) == 1.0, arr.ind = TRUE)
R> physical_points <- t(apply(edge_indexes - 1, MARGIN = 1,
+    sphere_image$TransformIndexToPhysicalPoint))
```

Finally, estimate sphere parameters using a least-squares fit.

```
R> A <- -2 * physical_points
R> A <- cbind(A, 1)
R> b <- -rowSums(physical_points^2)
R> x <- solve(qr(A, LAPACK = TRUE), b)
R> cat("The sphere's center is: ", x, "\n")
R> cat("The sphere's radius is: ", sqrt(x[1:3] %*% x[1:3] - x[4]), "\n")

The sphere's center is:  19.66792 -80.40793 9.260396 6925.624
The sphere's radius is:  3.52053
```

5.2. Intensity-based image registration

Registration is the process of computing the transformation that relates corresponding points in two different coordinate systems. It is a key component in many medical and non-medical applications, with a large number of algorithms described in the literature (Oliveira and Tavares 2014; Zitová and Flusser 2003).

Intensity-based image registration estimates the transformation that aligns images to each other based on their intensity values. The task is formulated as an optimization problem with the optimized function indicating how similar the two images are to each other after applying a given spatial transformation. As an image consists of a discrete set of intensity values at grid locations and the transformation is over a continuous domain, this formulation also requires the use of interpolation. The terminology which we use here refers to one image as the *fixed image* and the other, which is being transformed to match the fixed image, as the *moving image*. The transformation whose parameters we are optimizing maps points from the *fixed image* coordinate system to the *moving image* coordinate system. Registration is discussed in more detail in Fitzpatrick, Hill, and Maurer Jr (2000), Yaniv (2008), or Goshtasby (2005).

The four components that one needs to specify in order to configure the registration framework in **SimpleITK** are:

1. Transformation type – global domain (linear) transformations such as `AffineTransform` or `Euler3DTransform` are available, or local domain (nonlinear) transformations such as `BSplineTransform` and `DisplacementFieldTransform`. This choice defines the set of parameters whose values are optimized.

2. Similarity function – a model of the relationship between the intensity values of the two images such as `Correlation` for affine relationship and `MattesMutualInformation` for a stochastic relationship. The function is expected to attain a local minimum when the images are correctly aligned.

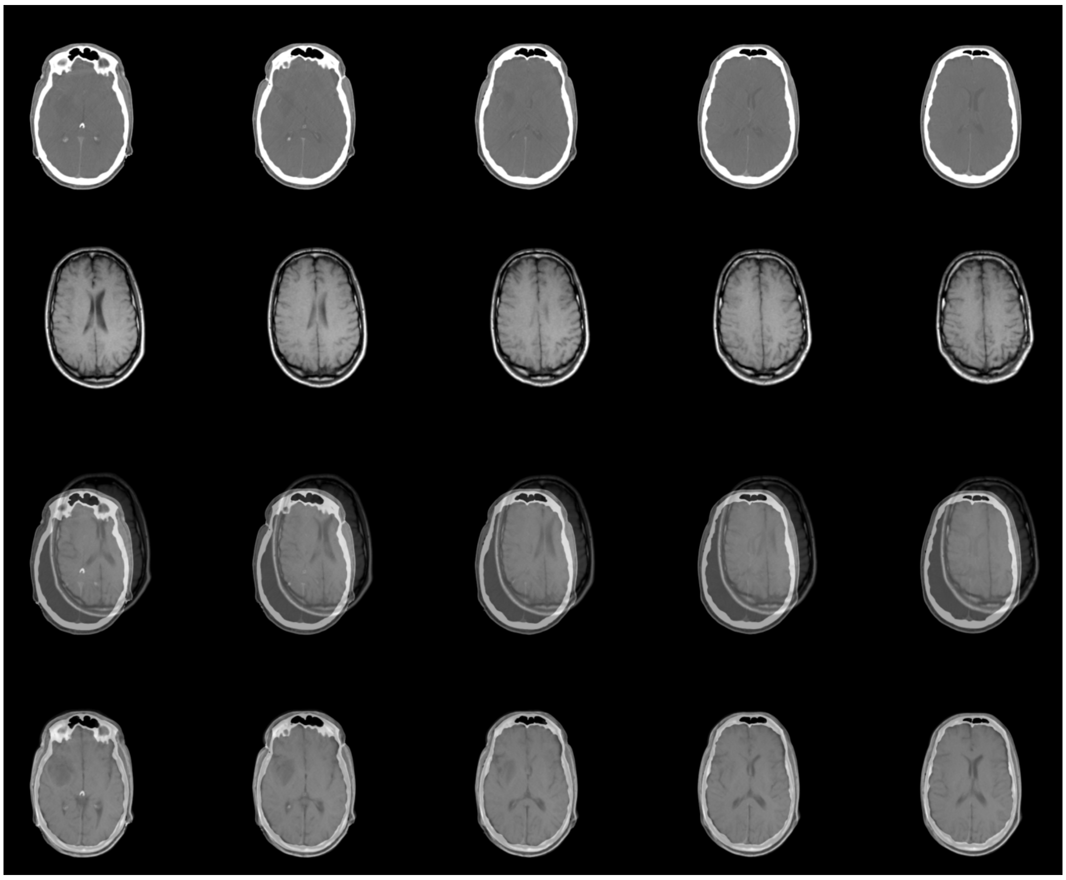


Figure 2: Five slices extracted from the center of each volume, from top to bottom: original CT image, our *fixed image*; original MR image, our *moving image*; fused image after initial spatial alignment of images; fused image after registration.

3. Interpolator – method to use for interpolating intensity values at non-grid locations, such as `sitkNearestNeighbor`, `sitkLinear` or `sitkHammingWindowedSinc`. This choice often reflects a compromise between accuracy and computational complexity with the most common choice being `sitkLinear`.

4. Optimizer – algorithm used to reach the optimum of the similarity function. These range from simple evaluation of the similarity using a discrete grid in the parameter space, `Exhaustive`, to a limited memory method with simple constraints, `L-BFGS-B`. There can be a complex relationship between an optimizer and parameters of a transform, especially when transform parameters have very different units (e.g., radians and millimeters).

An optional additional component, an *observer* function, can be added to report registration progress. One can add multiple functions to the same observed event or different functions for each observed event. Observer functions written in R are readily utilized.

In the following example we align a patient's cranial CT to their T1 MRI scan. This data is available online and is the training data set provided as part of the Retrospective Image Registration Evaluation project (Fitzpatrick 2016). The CT is a $512 \times 512 \times 29$ volume with a spacing of 0.65 × 0.65 × 4mm. The MRI is a 256 × 256 × 26 volume with a spacing of 1.25 × 1.25 × 4mm. Figure 2 shows five axial slices from the data at different phases of the registration process.

The specific component selections for the task at hand are as follows. As both images were obtained from the same patient, the transformation between them is expected to be rigid. We use a Euler angle-based parameterization of the transformation. As the intensities of the two modalities are related via a stochastic relationship we use mutual information as the similarity function. To reduce the computational burden of computing the similarity function we use 1% of the voxels, selected via simple random sampling, leading to slight differences in final transform between runs. The other available option is to obtain them using a regular grid overlaid onto the image. We use linear interpolation to obtain intensity values at non-grid locations. Finally, we use the basic gradient descent optimizer as our optimization method.

First we load the CT and T1 MR images:

```
R> library("SimpleITK")
R> fixed_image <- ReadImage("training_001_ct.mha", "sitkFloat32")
R> moving_image <- ReadImage("training_001_mr_T1.mha", "sitkFloat32")
```

Provide an initial alignment based on the centers of the two volumes:

```
R> initial_tx <- CenteredTransformInitializer(fixed_image,
+    moving_image, Euler3DTransform(), "GEOMETRY")
```

Create the observer functions. These will store similarity metric values for each iteration so that the progress can be visualized:

```
R> start_plot <- function() {
+    metric_values <<- c()
+  }
R> plot_values <- function(registration_method) {
+    metric_values <<- c(metric_values,
+      registration_method$GetMetricValue())
+  }
```

Create the registration object and attach observers:

```
R> reg <- ImageRegistrationMethod()
R> reg$AddCommand("sitkStartEvent", start_plot)
R> reg$AddCommand("sitkIterationEvent", function() plot_values(reg))
```

Configure the registration object and execute:

```
R> reg$SetMetricAsMattesMutualInformation(numberOfHistogramBins = 50)
R> reg$SetMetricSamplingStrategy("RANDOM")
R> reg$SetMetricSamplingPercentage(0.01)
R> reg$SetInterpolator("sitkLinear")
R> reg$SetOptimizerAsGradientDescent(learningRate = 1.0,
+    numberOfIterations = 100)
```

```
R> reg$SetOptimizerScalesFromPhysicalShift()
R> reg$SetInitialTransform(initial_tx, inPlace = FALSE)
R> final_tx <- reg$Execute(fixed_image, moving_image)
R> cat("The estimated transformation is:\n")
R> print(final_tx)
```

```
The estimated transformation is:
itk::simple::Transform
 CompositeTransform (0x54ab990)
   RTTI typeinfo:   itk::CompositeTransform<double, 3u>
   Reference Count: 1
   Modified Time: 689392
   Debug: Off
   Object Name:
   Observers:
     none
   Transforms in queue, from begin to end:
   >>>>>>>>>
   Euler3DTransform (0x64136c0)
     RTTI typeinfo:   itk::Euler3DTransform<double>
     Reference Count: 1
     Modified Time: 689383
     Debug: Off
     Object Name:
     Observers:
       none
     Matrix:
       0.999434 0.0324727 -0.00876782
       -0.0323064 0.999307 0.018489
       0.00936213 -0.0181952 0.999791
     Offset: [13.4809, -28.0635, -20.2838]
...
```

Two features of the registration framework that are specific to **ITK** and **SimpleITK** are the use of a so-called centered transform, `CenteredTransformInitializer`, and the automated estimation of parameter scales, `SetOptimizerScalesFromPhysicalShift`, for gradient-based optimizers. The centered transform performs rotation about the center of the fixed image, rather than the origin of the coordinate frame. The automated estimation of parameter scales deals with the complex relationship between transform terms with different units. For a comprehensive overview of the **ITK** registration framework we refer the interested reader to Avants, Tustison, Stauffer, Song, Wu, and Gee (2014).

Finally, even when the registration appears to have been performed successfully as validated by visual inspection (i.e., Figure 2) we always check the reason for the optimizer's termination. In our case we see below that we are likely dealing with premature termination, as the optimizer terminated because it reached the maximal number of iterations which we set. However the rate of improvement in similarity measure, illustrated in Figure 3 and based

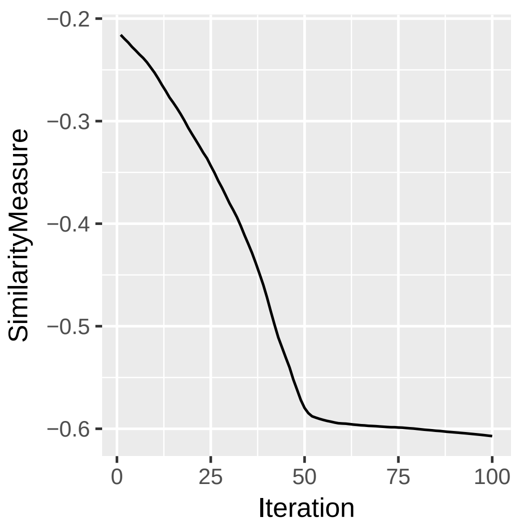


Figure 3: Similarity metric changes during rigid registration.

on information stored by the observer functions, decreased markedly after iteration 50. The solution is therefore unlikely to improve much with further iterations.

```
R> reg$GetOptimizerStopConditionDescription()

[1] "GradientDescentOptimizerv4Template:"
[2] "Maximum number of iterations (100) exceeded."
```

Rigid body registration, as illustrated in this example, is useful for aligning images of the same patient taken at similar times. This includes different modalities acquired during a study, such as different MR weightings, CT or PET as well as time series images, such as fMRI, where rigid body registration can be used to correct for patient movement.

5.3. Segmentation evaluation

Evaluation of segmentation algorithms applied to natural, medical, and biomedical images is most often done by comparing the output from the algorithm to those of human raters. It is common practice to derive an underlying reference segmentation by combining the annotations obtained from multiple raters. All raters can then be compared to the reference, which is useful when one rater is not known to be better than all others. Creating a combined reference is not straightforward, i.e., majority vote, when the raters are lay people such as when using crowd sourcing platforms as Amazon's mechanical Turk (Ipeirotis, Provost, and Wang 2010). Surprisingly, this is also not straightforward when the raters are domain experts such as radiologists interpreting medical images (Warfield, Zou, and Wells III 2004).

In this example we illustrate how to use the simultaneous truth and performance level estimation (STAPLE) algorithm (Warfield *et al.* 2004) to derive a reference segmentation. We then use this derived segmentation to evaluate the quality of the segmentation provided by each of our raters. As no single quality measure reflects the quality of the segmentation we generate a set of quality measures relating to overlap, over- and under-segmentation (false

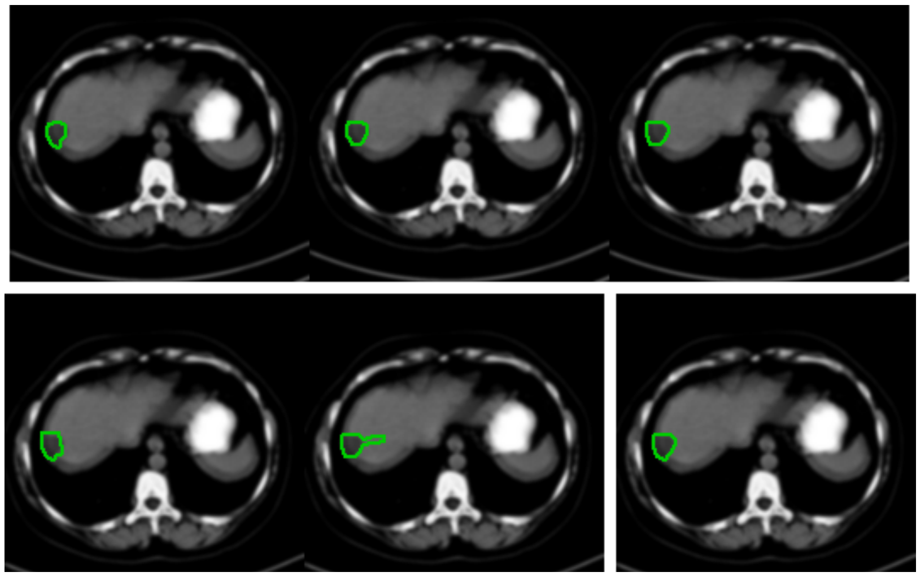


Figure 4: Deriving a reference segmentation from multiple raters using the STAPLE algorithm: (top row) manual segmentations performed by three radiologists; (bottom row) two additional segmentations derived from the expert segmentations to illustrate the effects of over-segmentation and segmentation with outliers. Last image is the derived reference segmentation obtained by the STAPLE algorithm.

positive, false negative), and distances between the boundaries of the segmented objects. The overlap and error scores between regions S and T are defined as follows:

- Dice: $2\frac{|S \cap T|}{|S| + |T|}$.
- Jaccard: $2\frac{|S \cap T|}{|S \cup T|}$.
- Volume similarity: $2\frac{|S| - |T|}{|S| + |T|}$.
- False negative: $\frac{|T \setminus S|}{|T|}$.
- False positive: $\frac{|S \setminus T|}{|S|}$.

Scores for boundary distances are based on summaries (max, mean, median and standard deviation) of the distance between the contour defined by S and the closest point on the contour of T. These measures characterize difference in outline. When comparing two large regions, a large maximum distance between boundaries may result in a very small difference in overlap scores, for example if caused by a single isolated voxel. Whether this distinction is important is application dependent.

In our case we use a single slice from an abdominal CT scan in which three radiologists segmented a liver tumor. We added two additional segmentations with intentional errors resulting in over-segmentation and segmentation which contains an outlying region. Figure 4 visually illustrates the inputs to the STAPLE algorithm and the derived reference segmentation.

We next look at the code used to generate the reference segmentation and how it is used to evaluate segmentations. We start by loading the segmentations provided by our raters.

```
R> image <- ReadImage("liver_tumor.mha")
R> segnames <- list.files(pattern = "liver_tumor_segmentation_.\\.mha",
+    path = "Code", full.names = TRUE)
R> names(segnames) <- gsub("liver_tumor_segmentation_(.+)\\.mha",
+    "rater_\\1", basename(segnames))
R> segmentations <- lapply(segnames, ReadImage)
```

We now generate the reference segmentation. Note that the input images are passed to STAPLE using a list.

```
R> foreground_value <- 1
R> threshold <- 0.95
R> reference_segmentation_STAPLE_probabilities <- STAPLE(segmentations,
+    foreground_value)
R> STAPLE_reference <- reference_segmentation_STAPLE_probabilities >
+    threshold
```

Using the derived reference segmentation we can compare how each of the raters agrees with our reference segmentation. Note that in practice you would compare a new rater or a segmentation algorithm's performance to the derived reference segmentation. Two common ways to perform this evaluation include computation of overlap and boundary distance scores.

Computing the overlap scores is straightforward, simply provide the two segmentations as input to the `LabelOverlapMeasuresImageFilter`, which is what we do in the utility function `compute_overlap_measures`.

Computing the boundary distance scores is slightly more involved. This is a two step process, where first an unsigned distance map from the reference data is generated. Then for each segmentation, the voxels on its boundary are labeled, `LabelContour`, and the intensity statistics is computed using the label as a mask and the distance map as the image for the `LabelIntensityStatisticsImageFilter`. This is implemented in the utility function `compute_surface_distance_measures`.

```
R> overlap_measures <- t(sapply(segmentations, compute_overlap_measures,
+    reference_segmentation = STAPLE_reference))
R> overlap_measures <- as.data.frame(overlap_measures)
R> overlap_measures$rater <- rownames(overlap_measures)
R> distance_map_filter <- SignedMaurerDistanceMapImageFilter()
R> distance_map_filter$SquaredDistanceOff()
R> STAPLE_reference_distance_map <-
+    abs(distance_map_filter$Execute(STAPLE_reference))
R> surface_distance_measures <- t(sapply(segmentations,
+    compute_surface_distance_measures,
+    reference_distance_map = STAPLE_reference_distance_map))
```

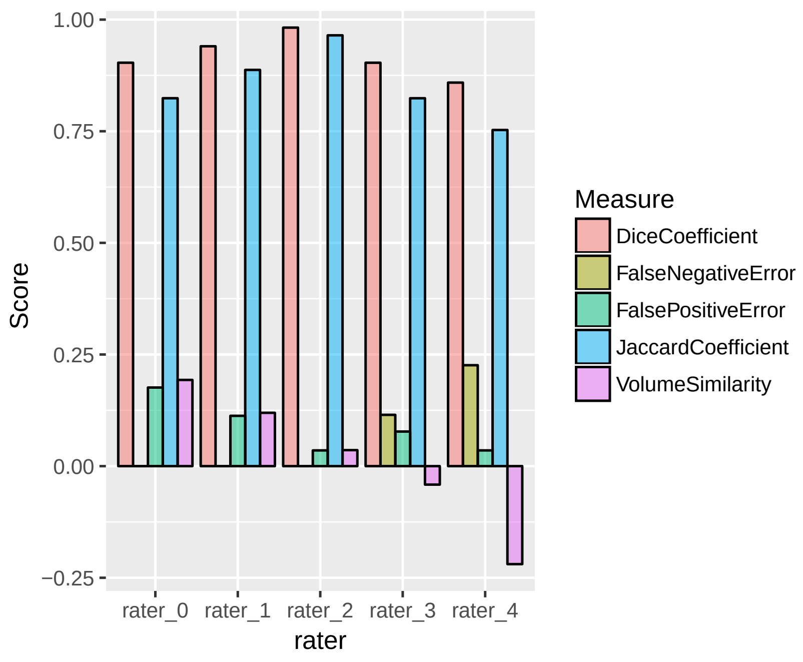


Figure 5: Comparison of raters using various overlap measures.

```
R> surface_distance_measures <- as.data.frame(surface_distance_measures)
R> surface_distance_measures$rater <- rownames(surface_distance_measures)
```

It is straightforward to visually compare raters using their overlap scores, as illustrated in the code below, with the results shown in Figure 5.

```
R> library("tidyr")
R> library("ggplot2")
R> overlap.gathered <- gather(overlap_measures, key = Measure,
+    value = Score, -rater)
R> ggplot(overlap.gathered,
+    aes(x = rater, y = Score, group = Measure, fill = Measure)) +
+    geom_bar(stat = "identity", position = "dodge",
+    colour = "black", alpha = 0.5)
```

Displaying the boundary distance data in table format, Table 1, is just as easy.

```
R> library("xtable")
R> sd <- surface_distance_measures
R> sd$rater <- NULL
R> xtable(sd, caption = "Surface distance measures",
+    label = "tab:surfdist", digits = 2)
```

5.4. Cell segmentation

Segmentation of cells in fluorescent microscopy is a relatively common image characterization task with variations that are dependent on the specifics of fluorescent markers for a given experiment. A typical procedure might include:

	Mean	Median	SD	Max
rater_0	0.33	0.65	0.49	1.41
rater_1	0.14	0.65	0.35	1.00
rater_2	0.05	0.65	0.23	1.00
rater_3	0.52	0.65	0.56	1.41
rater_4	2.82	0.65	4.07	12.17

Table 1: Surface distance measures.

- Histogram-based threshold estimation to produce a binary image.
- Cell splitting (separating touching cells) using distance transforms and watershed transform.
- Refinement of initial segmentation using information from other channels.
- Cell counting/characterization.

This example demonstrates the procedure on a 3 channel fluorescent microscopy image. The blue channel is a DNA marker (DAPI) that indicates all cells, the red channel is a marker of cell death (Ph3) while the green channel is a marker of cell proliferation (Ki67). A typical experiment might count the number of cells and measure size in the different states, where states are determined by presence of Ph3 and Ki67, various times after treatment with a drug candidate.

Cell segmentation and splitting

Histogram-based threshold estimation is performed by the `segChannel` function, listed below. It applies light smoothing followed by the Li threshold estimator (Li and Tam 1998), one of a range of threshold estimation options available in **SimpleITK**. A cell splitting procedure using distance transforms and a marker-based watershed (implemented by `segBlobs`, also listed below) was then applied to the resulting mask. Distance transforms replace each foreground pixel with the distance to the closest background pixel, producing a cone-shaped brightness profile for each circular object. Touching cells can then be separated using the peaks of the cones as markers in a watershed transform. A marker image is created by identifying peaks in the distance transform and applying a connected-component labeling. The inverted distance transform is used as the control image for the watershed transform.

The original image, provided in Nowell (2015), is illustrated in Figure 6 and processing stages are illustrated for an image subset (lower left part of original) in Figures 7 to 8.

```
R> segChannel <- function(dapi, dtsmooth = 3, osmooth = 0.5) {
+    dapi.smooth <- SmoothingRecursiveGaussian(dapi, osmooth)
+    th <- LiThresholdImageFilter()
+    th$SetOutsideValue(1)
+    th$SetInsideValue(0)
+    B <- th$Execute(dapi.smooth)
+    g <- splitBlobs(B, dtsmooth)
+    return(list(thresh = B, labels = g$labels, peaks = g$peaks,
+       dist = g$dist))
+   }
R> splitBlobs <- function(mask, smooth = 1) {
+     DT <- DanielssonDistanceMapImageFilter()
+     DT$UseImageSpacingOn()
+     distim <- DT$Execute(!mask)
+
+     distimS <- SmoothingRecursiveGaussian(distim, smooth, TRUE)
+     distim <- distimS * Cast(distim > 0, "sitkFloat32")
+
+     peakF <- RegionalMaximaImageFilter()
+     peakF$SetForegroundValue(1)
+     peakF$FullyConnectedOn()
+     peaks <- peakF$Execute(distim)
+
+     markers <- ConnectedComponent(peaks, TRUE)
+
+     WS <- MorphologicalWatershedFromMarkers(-1 * distim,
+       markers, TRUE, TRUE)
+     WS <- WS * Cast(distim > 0, WS$GetPixelID())
+     return(list(labels = WS, dist = distimS, peaks = peaks))
+   }
```

Load the data and place them into an red/green/blue image for display. The original is formatted as a 3D tif.

```
R> cntrl <- ReadImage("Control.tif")
R> red <- cntrl[, , 1]
R> green <- cntrl[, , 2]
R> blue <- cntrl[, , 3]
R> cntrl.colour <- Compose(red, green, blue)
R> dapi.cells <- segChannel(blue, 3)
R> ph3.cells <- segChannel(red, 3)
R> Ki67.cells <- segChannel(green, 3)
```

Refinement of segmentation

The Ph3 and Ki67 stains do not mark the entire cell nuclei. In some cases the marker is much smaller than the nucleus and there may be multiple markers per nucleus, leading to errors in counts or areas. We can, however, use the segmentation results from these channels to perform a geodesic reconstruction based on the DAPI segmentation results to reliably segment cells with those markers. The steps below mask the Ph3 and Ki67 segmentations using the DAPI segmentation and then apply a geodesic reconstruction that “grows” the Ph3/Ki67 cell segmentation to the size of the DAPI marked cell. Figure 8 illustrates the initial segmentation of the Ph3 and Ki67 channels while Figure 9 illustrates the results for the Ph3 channel.

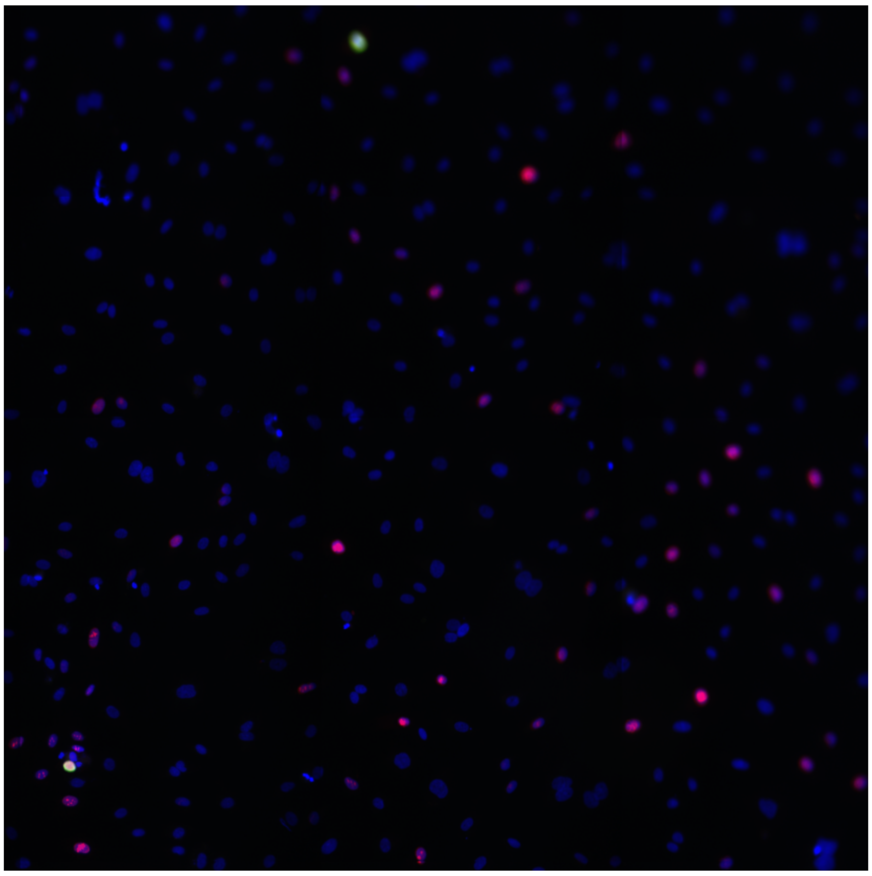
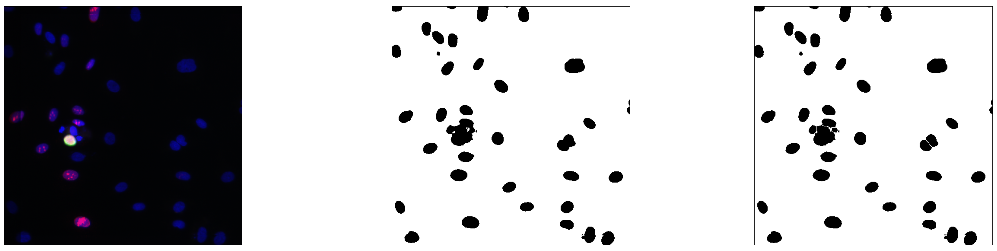


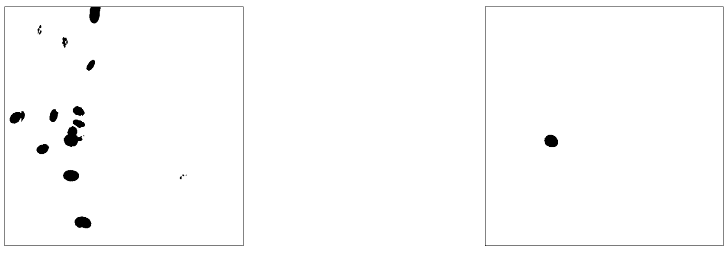
Figure 6: Confocal microscope image of cells stained with Ph3 (red), Ki67 (green) and DAPI (blue).



(a) Original 3 channel image. (b) Thresholded image. (c) Results of cell splitting.

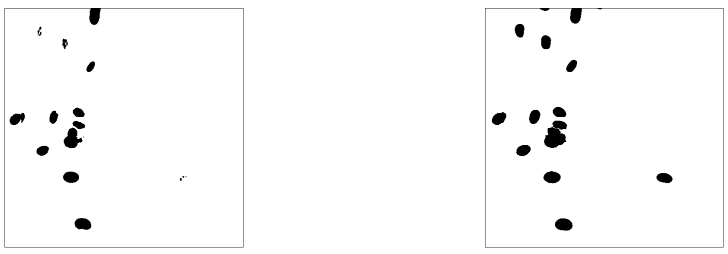
Figure 7: Stages of segmentation for the DAPI channel. Touching cells in the mid-right side of the image are separated by the splitting stage.

```
R> dapi.mask <- dapi.cells$labels != 0
R> ph3.markers <- ph3.cells$thresh * dapi.mask
R> Ki67.markers <- Ki67.cells$thresh * dapi.mask
```



(a) Ph3 channel segmentation. (b) Ki67 channel segmentation.

Figure 8: Segmentation using Li thresholding of Ph3 and Ki67 channels.



(a) Ph3 channel segmentation. (b) Ph3 channel refinement.

Figure 9: Ph3 segmentation refinement using geodesic reconstruction – note the increase in size of nuclei at the top and right of image.

```
R> ph3.recon <- BinaryReconstructionByDilation(ph3.markers, dapi.mask)
R> Ki67.recon <- BinaryReconstructionByDilation(Ki67.markers, dapi.mask)
```

Characterization and counting

Image segmentations can lead to quantitative measures such as counts and shape statistics (e.g., area, perimeter etc.). Such measures can be biased by edge effects, so it is useful to know whether the objects are touching the image edge. The classes used for these steps in **SimpleITK** are `ConnectedComponentImageFilter` and `LabelShapeStatisticsImageFilter`. The former produces a *labeled* image, in which each binary connected component is given a different integer voxel value. Label images are used in many segmentation contexts, including the cell splitting function illustrated earlier. The latter produces shape measures per connected component. The function below illustrates extraction of centroids, areas and edge touching measures. Cell counts are also available from the table dimensions.

```
R> getCellStats <- function(labelledim) {
+    StatsFilt <- LabelShapeStatisticsImageFilter()
+    StatsFilt$Execute(labelledim)
+
+    objs <- StatsFilt$GetNumberOfLabels()
+
```

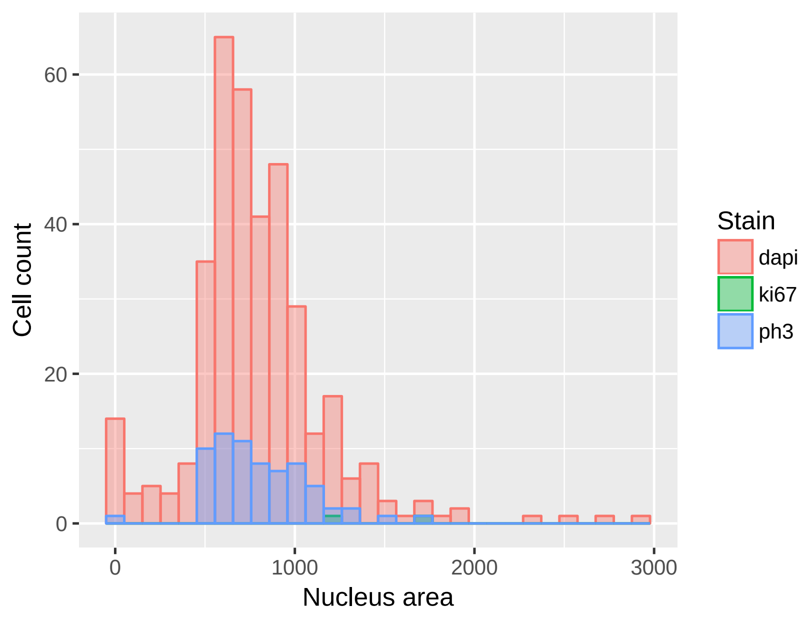


Figure 10: Histograms of cell nucleus area by stain type.

```
+    areas <- sapply(1:objs, StatsFilt$GetPhysicalSize)
+    boundarycontact <- sapply(1:objs, StatsFilt$GetNumberOfPixelsOnBorder)
+    centroid <- t(sapply(1:objs, StatsFilt$GetCentroid))
+
+    result <- data.frame(Area = areas,
+      TouchingImageBoundary = boundarycontact,
+      Cx = centroid[, 1], Cy = centroid[, 2])
+    return(result)
+  }
R> dapi.stats <- getCellStats(dapi.cells$labels)
R> head(dapi.stats)

  Area TouchingImageBoundary        Cx         Cy
1   18                    14  962.5000  0.2222222
2  722                    30  548.1745 11.1149584
3  296                    24  718.8446  5.1047297
4  794                     0  709.3489 22.0214106
5 1403                     0 1671.6251 23.3036351
6  775                     0 1315.4839 28.7987097
```

Using this data we can begin to visualize the properties of the cell population – for example distributions of cell areas – Figure 10. Note that the cell measures include information about which cells touch the image boundary, allowing easy removal to avoid biasing area statistics.

```
R> ph3.recon.labelled <- ConnectedComponent(ph3.recon)
R> Ki67.recon.labelled <- ConnectedComponent(Ki67.recon)
R> dapistats <- getCellStats(dapi.cells$labels)
```

```
R> ph3stats <- getCellStats(ph3.recon.labelled)
R> ki67stats <- getCellStats(Ki67.recon.labelled)
R> dapistats$Stain <- "dapi"
R> ph3stats$Stain <- "ph3"
R> ki67stats$Stain <- "ki67"
R> cellstats <- rbind(dapistats, ph3stats, ki67stats)
R> cellstats$Stain <- factor(cellstats$Stain)
R> cellstats.no.boundary <- subset(cellstats, TouchingImageBoundary == 0)
```

6. Discussion and conclusions

There are a large number of R packages with imaging capabilities, many of which are discussed in the CRAN task view on medical imaging (Whitcher 2018) and the recent special issue on "Magnetic Resonance Imaging in R" of this journal (Tabelow and Whitcher 2011). Some packages implement IO of standard or specialized image formats: **tiff** (Urbanek 2013b), **jpeg** (Urbanek 2014), **png** (Urbanek 2013a), **bmp** (Jefferis 2017), **pixmap** (Bivand, Leisch, and Maechler 2011), **R4dfp** (Barry and Snyder 2013), **readbitmap** (Jefferis 2014), **tractor.base** (Clayden 2018; Clayden, Maniega, Storkey, King, Bastin, and Clark 2011), **oro.nifti** (Whitcher, Schmid, and Thornton 2017; Whitcher, Schmid, and Thorton 2011), **oro.dicom** (Whitcher 2015; Whitcher *et al.* 2011) **neuroim** (Buchsbaum 2016). Such packages typically make image pixel data available as an R array and possibly provide access to file format specific image meta-data.

The advanced model fitting capabilities of R are used to implement a range of domain-specific algorithms for image data. Examples in the medical domain include **dpmixsim** (da Silva 2012) and **mritc** (Feng and Tierney 2015, 2011) for structural MRI, **dti** (Tabelow, Polzehl, and Anker 2016; Polzehl and Tabelow 2011) and **tractor.base** for diffusion MRI, and **AnalyzeFMRI** (Lafaye de Micheaux and Marchini 2018; Bordier, Dojat, and Micheaux 2011), **fmri** (Tabelow and Polzehl 2016b, 2011) for functional MRI and **DATforDCEMRI** (Ferl 2013, 2011) and **dcemriS4** (Whitcher, Schmid, and Thornton 2015; Whitcher and Schmid 2011) for dynamic contrast-enhanced MRI and **oasis** (Sweeney, Muschelli, and Taki Shinohara 2018) for lesion segmentation in multiple sclerosis. Diverse modalities are supported by other packages, such as atomic force microscopy by **AFM** (Beauvais, Liascukiene, and Landoulsi 2018), satellite imagery by **landsat** (Goslee 2012) and **ripa** (Frery and Perciano 2013), thermal imaging by **Thermimage** (Tattersall 2017), and more specific packages such as **measuRing** (Lara, Sierra, and Felipe Bravo 2018) which provides tools for characterizing growth rings in trees.

Other packages focus on specific techniques, such as texture analysis (**radiomics**; Carlson 2016), smoothing (**adimpro**; Tabelow and Polzehl 2016a), boundary detection (**BayesBD**; Li 2017), and registration (**NiftiReg**; Modat *et al.* 2010; and **antsR**; Avants, Kandel, Duda, Cook, and Tustison 2015).

Finally, some packages provide interfaces to external imaging applications, such as **fslr** (Muschelli 2018) and **RImageJ** (Francois, Grosjean, and Murrell 2015), interfacing to **FSL** and **ImageJ** (Schneider, Rasband, and Eliceiri 2012; Schindelin *et al.* 2012) respectively. R is acting as an advanced shell to run executables from those packages.

Although many of the packages above may have functionality that is applicable to multiple domains, they are not designed to be general purpose packages. Two exceptions are

imager (Barthelme 2017), which utilizes the **CImg** library to provide a range of filtering, thresholding and warping functions and **EBImage** (Pau, Fuchs, Sklyar, Boutros, and Huber 2010) which provides similar capabilities for cell imaging in microscopy. Both use R arrays to represent images. **SimpleITK** offers a much greater range of computational functions than either of these packages.

The **SimpleITK** package introduced in this article supports IO for a wide variety of image formats and provides a broad range of computational components to perform filtering, segmentation and registration. It uses the image data structures from **ITK**, which include a complete set of meta-data describing image and voxel geometry in world coordinates, and processing classes, which have been widely used for many years and have solid software engineering support to provide long-term maintainability.

SimpleITK provides an open source solution to complex image analysis tasks with facilities that rival many commercial offerings. The option of using **SimpleITK** with interpreter-based environments such as R allows a researcher or developer to quickly explore many combinations of computational strategies when working with images.

To follow the ongoing toolkit development go to: `https://github.com/SimpleITK/SimpleITK`. We hope that **SimpleITK** will be useful to anyone faced with the task of obtaining insights from images.

Acknowledgments

This work was supported by the Intramural Research Program of the U.S. National Institutes of Health, National Library of Medicine, the Murdoch Children's Research Institute, Melbourne, Australia and the Victorian Government's Operational Infrastructure Support Program.

References

Avants BB, Kandel BM, Duda JT, Cook PA, Tustison NJ (2015). ***antsR**: **ANTs** in R.* R package version 0.3.2, URL `http://stnava.github.io/ANTsR/`.

Avants BB, Tustison NJ, Song G, Cook PA, Klein A, Gee JC (2011). "A Reproducible Evaluation of **ANTs** Similarity Metric Performance in Brain Image Registration." *NeuroImage*, **54**(3), 2033–2044. `doi:10.1016/j.neuroimage.2010.09.025`.

Avants BB, Tustison NJ, Stauffer M, Song G, Wu B, Gee JC (2014). "The Insight ToolKit Image Registration Framework." *Frontiers in Neuroinformatics*, **8**(44). `doi:10.3389/fninf.2014.00044`.

Barry KP, Snyder AZ (2013). ***R4dfp**: **4dfp** MRI Image Read and Write Routines.* R package version 0.2-4, URL `http://CRAN.R-project.org/package=R4dfp`.

Barthelme S (2017). ***imager**: Image Processing Library Based on **CImg**.* R package version 0.40.2, URL `http://CRAN.R-project.org/package=imager`.

Beauvais M, Liascukiene I, Landoulsi J (2018). ***AFM**: Atomic Force Microscope Image Analysis.* R package version 1.2.4, URL `http://CRAN.R-project.org/package=AFM`.

Beazley DM, *et al.* (1996). "**SWIG**: An Easy to Use Tool for Integrating Scripting Languages with C and C++." In *Tcl/Tk Workshop*.

Bivand R, Leisch F, Maechler M (2011). ***pixmap**: Bitmap Images ("Pixel Maps")*. R package version 0.4-11, URL http://CRAN.R-project.org/package=pixmap.

Bordier C, Dojat M, Micheaux P (2011). "Temporal and Spatial Independent Component Analysis for fMRI Data Sets Embedded in the **AnalyzeFMRI** R Package." *Journal of Statistical Software*, **44**(9), 1–24. doi:10.18637/jss.v044.i09.

Buchsbaum BR (2016). ***neuroim**: Data Structures and Handling for Neuroimaging Data*. R package version 0.0.6, URL http://CRAN.R-project.org/package=neuroim.

Carlson J (2016). ***radiomics**: 'Radiomic' Image Processing Toolbox*. R package version 0.1.2, URL http://CRAN.R-project.org/package=radiomics.

Christiansen T, Foy BD, Orwant J, Wall L (2012). *Programming Perl*. 4th edition. O'Reilly & Associates, Inc., Sebastopol.

Clayden J (2018). ***tractor.base**: Read, Manipulate and Visualise Magnetic Resonance Images*. R package version 3.2.2, URL http://CRAN.R-project.org/package=tractor.base.

Clayden J, Maniega S, Storkey A, King M, Bastin M, Clark C (2011). "**TractoR**: Magnetic Resonance Imaging and Tractography with R." *Journal of Statistical Software*, **44**(8), 1–18. doi:10.18637/jss.v044.i08.

da Silva AF (2012). ***dpmixsim**: Dirichlet Process Mixture Model Simulation for Clustering and Image Segmentation*. R package version 0.0-8, URL http://CRAN.R-project.org/package=dpmixsim.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Enquobahrie A, Cheng P, Gary K, Ibáñez L, Gobbi D, Lindseth F, Yaniv Z, Aylward S, Jomier J, Cleary K (2007). "The Image-Guided Surgery Toolkit **IGSTK**: An Open Source C++ Software Toolkit." *Journal of Digital Imaging*, **20**(Suppl. 1), 21–33. doi:10.1007/s10278-007-9054-3.

Fedorov A, Beichel R, Kalpathy-Cramer J, Finet J, Fillion-Robin JC, Pujol S, Bauer C, Jennings D, Fennessy F, Sonka M, Buatti J, Aylward S, Miller JV, Pieper S, Kikinis R (2012). "**3D Slicer** as an Image Computing Platform for the Quantitative Imaging Network." *Magnetic Resonance Imaging*, **30**(9), 1323–1341. doi:10.1016/j.mri.2012.05.001.

Feng D, Tierney L (2011). "**mritc**: A Package for MRI Tissue Classification." *Journal of Statistical Software*, **44**(7), 1–20. doi:10.18637/jss.v044.i07.

Feng D, Tierney L (2015). ***mritc**: MRI Tissue Classification*. R package version 0.5-0, URL http://CRAN.R-project.org/package=mritc.

Ferl G (2011). "**DATforDCEMRI**: An R Package for Deconvolution Analysis and Visualization of DCE-MRI Data." *Journal of Statistical Software*, **44**(4), 1–18. doi:10.18637/jss.v044.i03.

Ferl GZ (2013). ***DATforDCEMRI**: Deconvolution Analysis Tool for Dynamic Contrast Enhanced MRI.* R package version 0.55, URL http://CRAN.R-project.org/package=DATforDCEMRI.

Fitzpatrick JM (2016). "Retrospective Image Registration Evaluation Project." Accessed on 2016-05-02, URL http://www.insight-journal.org/rire/.

Fitzpatrick JM, Hill DLG, Maurer Jr CR (2000). "Image Registration." In *Handbook of Medical Imaging, Volume 2. Medical Image Processing and Analysis*. SPIE Press.

Francois R, Grosjean P, Murrell P (2015). ***RImageJ**: R Bindings for ImageJ.* R package version 0.2-146, URL https://R-forge.R-project.org/R/?group_id=451.

Frery AC, Perciano T (2013). *Introduction to Image Processing Using R: Learning by Examples*. Springer-Verlag. doi:10.1007/978-1-4471-4950-7.

Goshtasby AA (2005). *2-D and 3-D Image Registration for Medical, Remote Sensing, and Industrial Applications*. John Wiley & Sons.

Goslee S (2012). ***landsat**: Radiometric and Topographic Correction of Satellite Imagery.* R package version 1.0.8, URL http://CRAN.R-project.org/package=landsat.

Ierusalimschy R (2016). *Programming in Lua*. 4th edition. Lua.org.

Inglada J, Christophe E (2009). "The **ORFEO** Toolbox Remote Sensing Image Processing Software." In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*.

Ipeirotis PG, Provost F, Wang J (2010). "Quality Management on Amazon Mechanical Turk." In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pp. 64–67.

Jefferis G (2014). ***readbitmap**: Simple Unified Interface to Read Bitmap Images (BMP, JPEG, PNG).* R package version 0.1-4, URL http://CRAN.R-project.org/package=readbitmap.

Jefferis G (2017). ***bmp**: Read Windows Bitmap (BMP) Images.* R package version 0.3, URL http://CRAN.R-project.org/package=bmp.

Johnson H, Harris G, Williams K (2007). "**BRAINSFit**: Mutual Information Registrations of Whole-Brain 3D Images, Using the Insight Toolkit." *Insight Journal*. July–December.

Johnson HJ, McCormick M, Ibáñez L, The Insight Software Consortium (2013). *The **ITK** Software Guide*, 3rd edition. URL http://www.itk.org/ItkSoftwareGuide.pdf.

Klein S, Staring M, Murphy K, Viergever MA, Pluim JPW (2010). "**elastix**: A Toolbox for Intensity-Based Medical Image Registration." *IEEE Transactions on Medical Imaging*, **29**(1), 196–205. doi:10.1109/tmi.2009.2035616.

Lafaye de Micheaux P, Marchini JL (2018). ***AnalyzeFMRI**: Functions for Analysis of fMRI Datasets Stored in the ANALYZE or NIFTI Format.* R package version 1.1-17, URL http://CRAN.R-project.org/package=AnalyzeFMRI.

Lara W, Sierra C, Felipe Bravo F (2018). ***measuRing**: Detection and Control of Tree-Ring Widths on Scanned Image Sections.* R package version 0.5, URL http://CRAN.R-project.org/package=measuRing.

Li CH, Tam PKS (1998). "An Iterative Algorithm for Minimum Cross Entropy Thresholding." *Pattern Recognition Letters*, **19**(8), 771–776. doi:10.1016/s0167-8655(98)00057-9.

Li M (2017). ***BayesBD**: Bayesian Boundary Detection in Images*. R package version 1.2, URL http://CRAN.R-project.org/package=BayesBD.

Lowekamp BC, Chen DT, Ibáñez L, Blezek D (2013). "The Design of **SimpleITK**." *Frontiers in Neuroinformatics*, **7**, 1–14. doi:10.3389/fninf.2013.00045.

Modat M, *et al.* (2010). "Fast Free-Form Deformation Using Graphics Processing Units." *Computer Methods and Programs in Biomedicine*, **98**(3), 278–284. doi:10.1016/j.cmpb.2009.09.002.

Mosaliganti KR, Gelas A, Megason SG (2013). "An Efficient, Scalable, and Adaptable Framework for Solving Generic Systems of Level-Set PDEs." *Frontiers in Neuroinformatics*, **7**, 1–35. doi:10.3389/fninf.2013.00035.

Muschelli J (2018). ***fslr**: Wrapper Functions for **FSL** (FMRIB Software Library) from Functional MRI of the Brain (FMRIB)*. R package version 2.17.3, URL http://CRAN.R-project.org/package=fslr.

Nowell C (2015). "**Fiji** Training Notes, 5.0." http://imagej.net/User_Guides.

Oliveira FPM, Tavares JMRS (2014). "Medical Image Registration: A Review." *Computer Methods in Biomechanics and Biomedical Engineering*, **17**(2), 73–93. doi:10.1080/10255842.2012.670855.

Pau G, Fuchs F, Sklyar O, Boutros M, Huber W (2010). "**EBImage** – An R Package for Image Processing with Applications to Cellular Phenotypes." *Bioinformatics*, **26**(7), 979–981. doi:10.1093/bioinformatics/btq046.

Polzehl J, Tabelow K (2011). "Beyond the Gaussian Model in Diffusion-Weighted Imaging: The Package **dti**." *Journal of Statistical Software*, **44**(12), 1–26. doi:10.18637/jss.v044.i12.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rossum GV, *et al.* (2011). *Python Programming Language*. URL https://www.python.org/.

Schindelin J, *et al.* (2012). "**Fiji**: An Open-Source Platform for Biological Image Analysis." *Nature Methods*, **9**(7), 676–682. doi:10.1038/nmeth.2019.

Schneider CA, Rasband WS, Eliceiri KW (2012). "NIH Image to **ImageJ**: 25 Years of Image Analysis." *Nature Methods*, **9**(7), 671–675. doi:10.1038/nmeth.2089.

Sweeney EM, Muschelli J, Taki Shinohara R (2018). ***oasis**: Multiple Sclerosis Lesion Segmentation Using Magnetic Resonance Imaging (MRI)*. R package version 3.0.4, URL http://CRAN.R-project.org/package=oasis.

Tabelow K, Polzehl J (2011). "Statistical Parametric Maps for Functional MRI Experiments in R: The Package **fmri**." *Journal of Statistical Software*, **44**(11), 1–21. doi:10.18637/jss.v044.i11.

Tabelow K, Polzehl J (2016a). ***adimpro**: Adaptive Smoothing of Digital Images.* R package version 0.8.2, URL `http://CRAN.R-project.org/package=adimpro`.

Tabelow K, Polzehl J (2016b). ***fmri**: Analysis of fMRI Experiments.* R package version 1.7-2, URL `http://CRAN.R-project.org/package=fmri`.

Tabelow K, Polzehl J, Anker F (2016). ***dti**: Analysis of Diffusion Weighted Imaging (DWI) Data.* R package version 1.2-6.1, URL `http://CRAN.R-project.org/package=dti`.

Tabelow K, Whitcher B (2011). "Special Volume on Magnetic Resonance Imaging in R." *Journal of Statistical Software*, **44**(1), 1–6. `doi:10.18637/jss.v044.i01`.

Tattersall GJ (2017). ***Thermimage**: Functions for Handling Thermal Images.* R package version 3.1.0, URL `http://CRAN.R-project.org/package=Thermimage`.

Tcl Core Team (2017). *Tcl: Tool Commander Language.* URL `http://www.tcl.tk/`.

The Insight Software Consortium (2016a). "Building **SimpleITK**." URL `https://simpleitk.readthedocs.io/en/master/Documentation/docs/source/building.html`.

The Insight Software Consortium (2016b). "**SimpleITK** Documentation." URL `https://itk.org/SimpleITKDoxygen/html`.

The Insight Software Consortium (2018). ***SimpleITK**: Bindings to **SimpleITK** Image Segmentation and Registration Toolkit.* R package version 1.1.0, URL `http://www.simpleitk.org/`.

Thomas D, Fowler C, Hunt A (2009). *Programming Ruby 1.9: The Pragmatic Programmer's Guide.* The Facets of Ruby. The Pragmatic Bookshelf, Raleigh, North Carolina.

Toussaint N, Souplet JC, Fillard P (2007). "**medINRIA**: Medical Image Navigation and Research Tool by INRIA." In *MICCAI Workshop on Interaction in Medical Image Analysis and Visualization.*

Urbanek S (2013a). ***png**: Read and Write PNG Images.* R package version 0.1-7, URL `http://CRAN.R-project.org/package=png`.

Urbanek S (2013b). ***tiff**: Read and Write TIFF Images.* R package version 0.1-5, URL `http://CRAN.R-project.org/package=tiff`.

Urbanek S (2014). ***jpeg**: Read and Write JPEG Images.* R package version 0.1-8, URL `http://CRAN.R-project.org/package=jpeg`.

Warfield SK, Zou KH, Wells III WM (2004). "Simultaneous Truth and Performance Level Estimation (STAPLE): An Algorithm for the Validation of Image Segmentation." *IEEE Transactions on Medical Imaging*, **23**(7), 903–921. `doi:10.1109/tmi.2004.828354`.

Whitcher B (2015). ***oro.dicom**: Rigorous – DICOM Input/Output.* R package version 0.5.0, URL `http://CRAN.R-project.org/package=oro.dicom`.

Whitcher B (2018). *CRAN Task View: Medical Image Analysis.* Version 2018-01-24, URL `https://CRAN.R-project.org/view=MedicalImaging`.

Whitcher B, Schmid V (2011). "Quantitative Analysis of Dynamic Contrast-Enhanced and Diffusion-Weighted Magnetic Resonance Imaging for Oncology in R." *Journal of Statistical Software*, **44**(5), 1–29. doi:10.18637/jss.v044.i05.

Whitcher B, Schmid V, Thornton A (2015). ***dcemriS4**: A Package for Image Analysis of DCE-MRI (S4 Implementation).* R package version 0.55, URL http://CRAN.R-project.org/package=dcemriS4.

Whitcher B, Schmid V, Thornton A (2017). ***oro.nifti**: Rigorous – NIfTI + ANALYZE + AFNI: Input/Output.* R package version 0.9-1, URL http://CRAN.R-project.org/package=oro.nifti.

Whitcher B, Schmid V, Thorton A (2011). "Working with the DICOM and NIfTI Data Standards in R." *Journal of Statistical Software*, **44**(6), 1–29. doi:10.18637/jss.v044.i06.

Wickham H, Hester J, Chang W (2018). ***devtools**: Tools to Make Developing R Packages Easier.* R package version 1.13.5, URL https://CRAN.R-project.org/package=devtools.

Wolf I, Vetter M, Wegner I, Böttger T, Nolden M, Schöbinger M, Hastenteufel M, Kunert T, Meinzer HP (2005). "The Medical Imaging Interaction Toolkit." *Medical Image Analysis*, **9**(6), 594–604. doi:10.1016/j.media.2005.04.005.

Yaniv Z (2008). "Rigid Registration." In T Peters, K Cleary (eds.), *Image-Guided Interventions Technology and Applications*, chapter 6. Springer-Verlag.

Yaniv Z (2009). "Localizing Spherical Fiducials in C-Arm Based Cone-Beam CT." *Medical Physics*, **36**(11), 4957–4966. doi:10.1118/1.3233684.

Yaniv Z, Lowekamp BC, Johnson HJ, Beare R (2018). "**SimpleITK** Image-Analysis Notebooks: a Collaborative Environment for Education and Reproducible Research." *Journal of Digital Imaging*, **31**(3), 290–303. doi:10.1007/s10278-017-0037-8.

Yoo TS, Ackerman MJ, Lorensen WE, Schroeder W, Chalana V, Aylward S, Metaxas D, Whitaker R (2002). "Engineering and Algorithm Design for an Image Processing API: A Technical Report on **ITK** – The Insight Toolkit." *Studies in Health Technology and Informatics*, **85**, 586–592. doi:10.3233/978-1-60750-929-5-586.

Yushkevich PA, Piven J, Cody Hazlett H, Gimpel Smith R, Ho S, Gee JC, Gerig G (2006). "User-Guided 3D Active Contour Segmentation of Anatomical Structures: Significantly Improved Efficiency and Reliability." *NeuroImage*, **31**(3), 1116–1128. doi:10.1016/j.neuroimage.2006.01.015.

Zitová B, Flusser J (2003). "Image Registration Methods: A Survey." *Image and Vision Computing*, **21**(11), 977–1000. doi:10.1016/s0262-8856(03)00137-9.

Affiliation:

Richard Beare
Monash University
Department of Medicin
Monash Medical Centre
Clayton, Melbourne, Australia, 3168
E-mail: Richard.Beare@monash.edu

Bradley Lowekamp, Ziv Yaniv
National Institutes of Health,
Office of High Performance Computing and Communications
National Library of Medicine
8600 Rockville Pike
Bethesda, MD, 20894, United States of America
E-mail: blowekamp@mail.nih.gov, zivyaniv@nih.gov

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
August 2018, Volume 86, Issue 8 *Submitted:* 2016-06-24
doi:10.18637/jss.v086.i08 *Accepted:* 2017-08-07