Package 'RNiftyReg'

September 12, 2020

Version 2.7.0

Date 2020-09-10

Title Image Registration Using the 'NiftyReg' Library

Maintainer Jon Clayden <code@clayden.org>

Imports Rcpp, RNifti, ore

Suggests jpeg, loder, mmand, testthat, covr

LinkingTo Rcpp, RcppEigen, RNifti

Description Provides an 'R' interface to the 'NiftyReg' image registration tools <https://github.com/KCL-BMEIS/niftyreg>. Linear and nonlinear registration are supported, in two and three dimensions.

License GPL-2

URL https://github.com/jonclayden/RNiftyReg

BugReports https://github.com/jonclayden/RNiftyReg/issues

Encoding UTF-8

RoxygenNote 7.1.0

NeedsCompilation yes

Author Jon Clayden [cre, aut] (<https://orcid.org/0000-0002-6608-0619>),
Marc Modat [aut],
Benoit Presles [aut],
Thanasis Anthopoulos [aut],
Pankaj Daga [aut]

Repository CRAN

Date/Publication 2020-09-12 05:51:01 UTC

R topics documented:

`applyTransform` *Apply a precomputed transformation*

Description

This function allows a precomputed transformation to be applied to a new image or set of points.

Usage

```
applyTransform(transform, x, interpolation = 3L, nearest = FALSE,
  internal = FALSE)
```

Arguments

`transform`	A transform, possibly obtained from `forward` or `reverse`.
`x`	A numeric vector, representing a pixel/voxel location in source space, or a matrix with rows representing such points, or an image with the same dimensions as the original source image.
`interpolation`	A single integer specifying the type of interpolation to be applied to the final resampled image. May be 0 (nearest neighbour), 1 (trilinear) or 3 (cubic spline). No other values are valid.
`nearest`	Logical value: if `TRUE` and `x` contains points, the nearest voxel centre location in target space will be returned. Otherwise a more precise subvoxel location will be given.
`internal`	If `FALSE`, the default, the returned image will be returned as a standard R array. If `TRUE`, it will instead be an object of class `"internalImage"`, containing only basic metadata and a C-level pointer to the full image. (See also `readNifti`.) This can occasionally be useful to save memory.

Details

Points may be transformed from source to target space exactly under an affine transformation, but nonlinear transformation is inexact. Its accuracy will depend to some extent on the density of the control point grid and the geometry of the deformation in the vicinity of the points of interest. Nevertheless, it should be quite sufficient for most purposes.

The method is to first convert the control points to a deformation field (cf. `deformationField`), which encodes the location of each target space voxel in the source space. The target voxel closest to the requested location is found by searching through this deformation field, and returned if `nearest` is `TRUE` or it coincides exactly with the requested location. Otherwise, a block of four voxels in each dimension around the point of interest is extracted from the deformation field, and the final location is estimated by local cubic spline regression.

Value

A resampled image or matrix of transformed points.

Author(s)

Jon Clayden <code@clayden.org>

See Also

`niftyreg.linear`, `niftyreg.nonlinear`

`buildAffine` *Build an affine matrix up from its constituent transformations*

Description

This function does the opposite to `decomposeAffine`, building up an affine matrix from its components. It can be useful for testing, or for rescaling images.

Usage

```
buildAffine(translation = c(0, 0, 0), scales = c(1, 1, 1), skews = c(0,
  0, 0), angles = c(0, 0, 0), source = NULL, target = NULL,
  anchor = c("none", "origin", "centre", "center"))
```

Arguments

`translation`	Translations along each axis, in `pixunits` units. May also be a list, such as that produced by `decomposeAffine`, with elements for translation, scales, skews and angles.
`scales`	Scale factors along each axis.
`skews`	Skews in the XY, XZ and YZ planes.

angles	Roll, pitch and yaw rotation angles, in radians. If source is two-dimensional, a single angle will be interpreted as being in the plane as expected.
source	The source image for the transformation (required).
target	The target image for the transformation. If NULL (the default), it will be equal to source, or a rescaled version of it if any of the scales are not 1. In the latter case the scales will be reset back to 1 to produce the right effect.
anchor	The fixed point for the transformation. Setting this parameter to a value other than "none" will override the translation parameter, with the final translation set to ensure that the requested point remains in the same place after transformation.

Value

A 4x4 affine matrix representing the composite transformation. Note that NiftyReg affines logically transform backwards, from target to source space, so the matrix may be the inverse of what is expected.

Author(s)

Jon Clayden <code@clayden.org>

See Also

decomposeAffine, isAffine

---

composeTransforms *Compose transformations*

Description

Compute the composition of two or more transforms, the single transform that combines their effects in order.

Usage

```
composeTransforms(...)
```

Arguments

...	Affine or nonlinear transforms, possibly obtained from forward or reverse.

Value

The composed transform. If all arguments are affines then the result will also be an affine; otherwise it will be a deformation field.

Note

The source image for the composed transform is generally the source image from the first transform, and the target is the target image from the second transform. However, the target image attached to half transforms (as calculated by `halfTransform`) generally has a modified xform, compared to the original target. Therefore, composing a half transform with itself may not be exactly equivalent to the original.

Author(s)

Jon Clayden <code@clayden.org>

See Also

`niftyreg.linear`, `niftyreg.nonlinear`, `deformationField`

`decomposeAffine` *Decompose an affine matrix into its constituent transformations*

Description

An affine matrix is composed of translation, scale, skew and rotation transformations. This function extracts these components, after first inverting the matrix so that it transforms from source to target space.

Usage

```
decomposeAffine(affine)
```

Arguments

`affine`	A 4x4 matrix representing an affine transformation matrix.

Value

A list with components:

scaleMatrix A 3x3 matrix representing only the scale operation embodied in the full affine transformation.

skewMatrix A 3x3 matrix representing only the skew operation embodied in the full affine transformation.

rotationMatrix A 3x3 matrix representing only the rotation operation embodied in the full affine transformation.

translation A length-3 named numeric vector representing the translations (in `pixunits` units) in each of the X, Y and Z directions.

scales A length-3 named numeric vector representing the scale factors in each of the X, Y and Z directions. Scale factors of 1 represent no effect.

skews A length-3 named numeric vector representing the skews in each of the XY, XZ and YZ planes.

angles A length-3 named numeric vector representing the rotation angles (in radians) about each of the X, Y and Z directions, i.e., roll, pitch and yaw.

Note

The decomposition is not perfect, and there is one particular degenerate case when the pitch angle is very close to `pi/2` radians, known as "Gimbal lock". In this case the yaw angle is arbitrarily set to zero.

Affine matrices embodying rigid-body transformations include only 6 degrees of freedom, rather than the full 12, so skews will always be zero and scales will always be unity (to within rounding error). Likewise, affine matrices derived from 2D registration will not include components relating to the Z direction.

Author(s)

Jon Clayden <code@clayden.org>

See Also

`buildAffine`, `isAffine`

`deformationField` *Calculate the deformation field for a transformation*

Description

This function is used to calculate the deformation field corresponding to a specified linear or nonlinear transformation. The deformation field gives the location in source image space corresponding to the centre of each voxel in target space. It is used as a common form for linear and nonlinear transformations, and allows them to be visualised.

Usage

```
deformationField(transform, jacobian = TRUE)
```

Arguments

`transform`	A transform, possibly obtained from `forward` or `reverse`.
`jacobian`	A logical value: if `TRUE`, a Jacobian determinant map is also calculated and returned in an attribute.

Value

An `"internalImage"` representing the deformation field. If requested, the Jacobian map is stored in an attribute, which can be extracted using the `jacobian` accessor function.

Author(s)

Jon Clayden <code@clayden.org>

See Also

niftyreg.linear, niftyreg.nonlinear

---

forward *Extract forward and reverse transformations*

Description

These functions extract forward and reverse transformations in a form compatible with applyTransform and other functions. They are (S3) generic, but only methods for "niftyreg" objects currently exist.

Usage

```
forward(object, ...)

## S3 method for class 'niftyreg'
forward(object, i = 1, ...)

reverse(object, ...)

## S3 method for class 'niftyreg'
reverse(object, i = 1, ...)
```

Arguments

object	An R object.
...	Additional arguments. Not currently used.
i	The transformation number to extract. There will only be more than one in the case of multiple registration.

Value

A transformation object, an image or affine matrix, with suitable attributes giving pointers to source and target images. If there is no transformation information in the object then NULL is returned.

Author(s)

Jon Clayden <code@clayden.org>

See Also

niftyreg, applyTransform

halfTransform *Calculate a half transformation*

Description

This function calculates the half-way transformation corresponding to its argument. Applying this transformation results in points or images in a space halfway between the original source and target images, which can be a useful common space in some applications.

Usage

```
halfTransform(transform)
```

Arguments

transform	A transform, possibly obtained from forward or reverse.

Value

The half-way transform, in a similar format to transform.

Author(s)

Jon Clayden <code@clayden.org>

See Also

niftyreg.linear, niftyreg.nonlinear

invertAffine *Invert an affine matrix*

Description

This function is used to invert an affine matrix. It is a wrapper around solve, which additionally sets appropriate attributes.

Usage

```
invertAffine(affine)
```

Arguments

affine	An existing 4x4 affine matrix.

Value

The inverted affine matrix.

Author(s)

Jon Clayden <code@clayden.org>

See Also

solve

Examples

```
affine <- readAffine(system.file("extdata","affine.txt",package="RNiftyReg"))
print(affine)
print(invertAffine(affine))
```

---

isAffine *Create, test for and print affine matrices*

Description

isAffine returns a logical value indicating whether its argument is, or resembles, a 4x4 affine matrix. asAffine converts a suitable matrix to the affine class, attaching the source and target images as attributes. Affine transformations are a class of linear transformations which preserve points, straight lines and planes, and may consist of a combination of rotation, translation, scale and skew operations.

Usage

```
isAffine(object, strict = FALSE)

asAffine(object, source = NULL, target = NULL)

## S3 method for class 'affine'
print(x, ...)
```

Arguments

object	An R object.
strict	If TRUE, this function just tests whether the object is of class "affine". Otherwise it also tests for an affine-like 4x4 matrix.
source, target	Source and target images for the transformation.
x	An "affine" object.
...	Additional parameters to methods. Currently unused.

Details

NiftyReg's convention is for affine matrices to transform world coordinates (in the sense of `voxelToWorld`) from TARGET to SOURCE space, although transforms are logically applied the other way.

Value

A logical value, which is `TRUE` if `object` appears to be an affine matrix.

Note

2D affines are a subset of 3D affines, and are stored in a 4x4 matrix for internal consistency, even though a 3x3 matrix would suffice.

Author(s)

Jon Clayden <code@clayden.org>

`isImage` *Test whether an object represents an image*

Description

This function tried to determine whether an object is an image that the package knows how to handle. If its class is `"nifti"`, `"niftiImage"`, `"internalImage"` or `"MriImage"`, then the result is always `TRUE`. Likewise if it has an internal image pointer. If it has no `dim` attribute, or looks like an affine matrix, then the result is `FALSE`. Otherwise the value of the `unsure` argument is returned.

Usage

```
isImage(object, unsure = NA)
```

Arguments

`object`	An R object.
`unsure`	The value to return if the function can't tell whether or not the `object` is an image.

Author(s)

Jon Clayden <code@clayden.org>

jacobian *Extract a Jacobian determinant map*

Description

This function extracts the Jacobian determinant map associated with a deformation field.

Usage

```
jacobian(x)
```

Arguments

x	An R object, probably a deformation field.

Author(s)

Jon Clayden <code@clayden.org>

See Also

deformationField

niftyreg *Two and three dimensional image registration*

Description

The niftyreg function performs linear or nonlinear registration for two and three dimensional images. 4D images may also be registered volumewise to a 3D image, or 3D images slicewise to a 2D image. This function is a common wrapper for niftyreg.linear and niftyreg.nonlinear.

Usage

```
niftyreg(source, target, scope = c("affine", "rigid", "nonlinear"),
  init = NULL, sourceMask = NULL, targetMask = NULL, symmetric = TRUE,
  interpolation = 3L, estimateOnly = FALSE, sequentialInit = FALSE,
  internal = NA, precision = c("double", "single"),
  threads = getOption("RNiftyReg.threads"), ...)

## S3 method for class 'niftyreg'
asNifti(x, ...)

## S3 method for class 'niftyreg'
as.array(x, ...)
```

Arguments

source	The source image, an object of class "nifti" or "internalImage", or a plain array, or a NIfTI-1 filename. Must have 2, 3 or 4 dimensions.
target	The target image, an object of class "nifti" or "internalImage", or a plain array, or a NIfTI-1 filename. Must have 2 or 3 dimensions.
scope	A string describing the scope, or number of degrees of freedom (DOF), of the registration. The currently supported values are "affine" (12 DOF), "rigid" (6 DOF) or "nonlinear" (high DOF, with the exact number depending on the image sizes).
init	Transformation(s) to be used for initialisation, which may be NULL, for no initialisation, or an affine matrix or control point image (nonlinear only). For multiple registration, where the source image has one more dimension than the target, this may also be a list whose components are likewise NULL or a suitable initial transform.
sourceMask	An optional mask image in source space, whose nonzero region will be taken as the region of interest for the registration. Ignored when symmetric is FALSE.
targetMask	An optional mask image in target space, whose nonzero region will be taken as the region of interest for the registration.
symmetric	Logical value. Should forward and reverse transformations be estimated simultaneously?
interpolation	A single integer specifying the type of interpolation to be applied to the final resampled image. May be 0 (nearest neighbour), 1 (trilinear) or 3 (cubic spline). No other values are valid.
estimateOnly	Logical value: if TRUE, transformations will be estimated, but images will not be resampled.
sequentialInit	If TRUE and source has higher dimensionality than target, transformations which are not explicitly initialised will begin from the result of the previous registration.
internal	If NA, the default, the final resampled image will be returned as a standard R array, but control point maps will be objects of class "internalImage", containing only basic metadata and a C-level pointer to the full image. (See also readNifti.) If TRUE, all image-type objects in the result will be internal images; if FALSE, they will all be R arrays. The default is fine for most purposes, but using TRUE may save memory, while using FALSE can be necessary if there is a chance that external pointers will be invalidated, for example when returning from worker threads.
precision	Working precision for the registration. Using single- precision may be desirable to save memory when coregistering large images.
threads	For OpenMP-capable builds of the package, the maximum number of threads to use.
...	Further arguments to niftyreg.linear or niftyreg.nonlinear.
x	A "niftyreg" object.

Value

A list of class "niftyreg" with components:

image An array or internal image representing the registered and resampled source image in the space of the target image. This element is NULL if the estimateOnly parameter is TRUE.

forwardTransforms A list of (linear or nonlinear) transformations from source to target space.

reverseTransforms A list of (linear or nonlinear) transformations from target to source space.

iterations A list of integer vectors, giving the number of iterations completed at each "level" of the algorithm. Note that for the first level of the linear algorithm specifically, twice the specified number of iterations is allowed.

source An internal representation of the source image for each registration.

target An internal representation of the target image.

The as.array method for this class returns the image element.

Note

If substantial parts of the target image are zero-valued, for example because the target image has been brain-extracted, it can be useful to pass it as a target mask as well as the target image, viz. niftyreg(source,target,targetMask=target).

Author(s)

Jon Clayden <code@clayden.org>

References

Please see niftyreg.linear or niftyreg.nonlinear for references relating to each type of registration.

See Also

niftyreg.linear and niftyreg.nonlinear, which do most of the work. Also, forward and reverse to extract transformations, and applyTransform to apply them to new images or points.

Examples

```
## Not run:
source <- readNifti(system.file("extdata", "epi_t2.nii.gz",
  package="RNiftyReg"))
target <- readNifti(system.file("extdata", "flash_t1.nii.gz",
  package="RNiftyReg"))

result <- niftyreg(source, target, scope="affine")

## End(Not run)
```

`niftyreg.linear` *Two and three dimensional linear image registration*

Description

The `niftyreg.linear` function performs linear registration for two and three dimensional images. 4D images may also be registered volumewise to a 3D image, or 3D images slicewise to a 2D image. Rigid-body (6 degrees of freedom) and affine (12 degrees of freedom) registration can currently be performed.

Usage

```
niftyreg.linear(source, target, scope = c("affine", "rigid"), init = NULL,
  sourceMask = NULL, targetMask = NULL, symmetric = TRUE, nLevels = 3L,
  maxIterations = 5L, useBlockPercentage = 50L, interpolation = 3L,
  verbose = FALSE, estimateOnly = FALSE, sequentialInit = FALSE,
  internal = NA, precision = c("double", "single"),
  threads = getOption("RNiftyReg.threads"))
```

Arguments

`source`	The source image, an object of class `"nifti"` or `"internalImage"`, or a plain array, or a NIfTI-1 filename. Must have 2, 3 or 4 dimensions.
`target`	The target image, an object of class `"nifti"` or `"internalImage"`, or a plain array, or a NIfTI-1 filename. Must have 2 or 3 dimensions.
`scope`	A string describing the scope, or number of degrees of freedom (DOF), of the registration. The currently supported values are `"affine"` (12 DOF), `"rigid"` (6 DOF) or `"nonlinear"` (high DOF, with the exact number depending on the image sizes).
`init`	Transformation(s) to be used for initialisation, which may be `NULL`, for no initialisation, or an affine matrix or control point image (nonlinear only). For multiple registration, where the source image has one more dimension than the target, this may also be a list whose components are likewise `NULL` or a suitable initial transform.
`sourceMask`	An optional mask image in source space, whose nonzero region will be taken as the region of interest for the registration. Ignored when `symmetric` is `FALSE`.
`targetMask`	An optional mask image in target space, whose nonzero region will be taken as the region of interest for the registration.
`symmetric`	Logical value. Should forward and reverse transformations be estimated simultaneously?
`nLevels`	A single integer specifying the number of levels of the algorithm that should be applied. If zero, no optimisation will be performed, and the final affine matrix will be the same as its initialisation value.

`maxIterations` A single integer specifying the maximum number of iterations to be used within each level. Fewer iterations may be used if a convergence test deems the process to have completed.

`useBlockPercentage` A single integer giving the percentage of blocks to use for calculating correspondence at each step of the algorithm. The blocks with the highest intensity variance will be chosen.

`interpolation` A single integer specifying the type of interpolation to be applied to the final resampled image. May be 0 (nearest neighbour), 1 (trilinear) or 3 (cubic spline). No other values are valid.

`verbose` A single logical value: if `TRUE`, the code will give some feedback on its progress; otherwise, nothing will be output while the algorithm runs. Run time can be seconds or more, depending on the size and dimensionality of the images.

`estimateOnly` Logical value: if `TRUE`, transformations will be estimated, but images will not be resampled.

`sequentialInit` If `TRUE` and `source` has higher dimensionality than `target`, transformations which are not explicitly initialised will begin from the result of the previous registration.

`internal` If `NA`, the default, the final resampled image will be returned as a standard R array, but control point maps will be objects of class `"internalImage"`, containing only basic metadata and a C-level pointer to the full image. (See also `readNifti`.) If `TRUE`, all image-type objects in the result will be internal images; if `FALSE`, they will all be R arrays. The default is fine for most purposes, but using `TRUE` may save memory, while using `FALSE` can be necessary if there is a chance that external pointers will be invalidated, for example when returning from worker threads.

`precision` Working precision for the registration. Using single- precision may be desirable to save memory when coregistering large images.

`threads` For OpenMP-capable builds of the package, the maximum number of threads to use.

Details

This function performs the dual operations of finding a transformation to optimise image alignment, and resampling the source image into the space of the target image.

The algorithm is based on a block-matching approach and Least Trimmed Squares (LTS) fitting. Firstly, the block matching provides a set of corresponding points between a target and a source image. Secondly, using this set of corresponding points, the best rigid or affine transformation is evaluated. This two-step loop is repeated until convergence to the best transformation is achieved.

In the NiftyReg implementation, normalised cross-correlation between the target and source blocks is used to evaluate correspondence. The block width is constant and has been set to 4 voxels. A coarse-to-fine approach is used, where the registration is first performed on down-sampled images (using a Gaussian filter to resample images), and finally performed on full resolution images.

The source image may have 2, 3 or 4 dimensions, and the target 2 or 3. The dimensionality of the target image determines whether 2D or 3D registration is applied, and source images with one more

dimension than the target (i.e. 4D to 3D, or 3D to 2D) will be registered volumewise or slicewise, as appropriate. In the latter case the last dimension of the resulting image is taken from the source image, while all other dimensions come from the target. One affine matrix is returned for each registration performed.

Value

See niftyreg.

Author(s)

Jon Clayden <code@clayden.org>

References

The algorithm used by this function is described in the following publication.

M. Modat, D.M. Cash, P. Daga, G.P. Winston, J.S. Duncan & S. Ourselin (2014). Global image registration using a symmetric block-matching approach. Journal of Medical Imaging 1(2):024003.

See Also

niftyreg, which can be used as an interface to this function, and niftyreg.nonlinear for non-linear registration. Also, forward and reverse to extract transformations, and applyTransform to apply them to new images or points.

---

niftyreg.nonlinear *Two and three dimensional nonlinear image registration*

Description

The niftyreg.nonlinear function performs nonlinear registration for two and three dimensional images. 4D images may also be registered volumewise to a 3D image, or 3D images slicewise to a 2D image. The warping is based on free-form deformations, parameterised using an image of control points.

Usage

```
niftyreg.nonlinear(source, target, init = NULL, sourceMask = NULL,
  targetMask = NULL, symmetric = TRUE, nLevels = 3L,
  maxIterations = 150L, nBins = 64L, bendingEnergyWeight = 0.001,
  linearEnergyWeight = 0.01, jacobianWeight = 0, finalSpacing = c(5, 5,
  5), spacingUnit = c("voxel", "world"), interpolation = 3L,
  verbose = FALSE, estimateOnly = FALSE, sequentialInit = FALSE,
  internal = NA, precision = c("double", "single"),
  threads = getOption("RNiftyReg.threads"))
```

Arguments

source	The source image, an object of class "nifti" or "internalImage", or a plain array, or a NIfTI-1 filename. Must have 2, 3 or 4 dimensions.
target	The target image, an object of class "nifti" or "internalImage", or a plain array, or a NIfTI-1 filename. Must have 2 or 3 dimensions.
init	Transformation(s) to be used for initialisation, which may be NULL, for no initialisation, or an affine matrix or control point image (nonlinear only). For multiple registration, where the source image has one more dimension than the target, this may also be a list whose components are likewise NULL or a suitable initial transform.
sourceMask	An optional mask image in source space, whose nonzero region will be taken as the region of interest for the registration. Ignored when symmetric is FALSE.
targetMask	An optional mask image in target space, whose nonzero region will be taken as the region of interest for the registration.
symmetric	Logical value. Should forward and reverse transformations be estimated simultaneously?
nLevels	A single integer specifying the number of levels of the algorithm that should be applied. If zero, no optimisation will be performed, and the final control-point image will be the same as its initialisation value.
maxIterations	A single integer specifying the maximum number of iterations to be used within each level. Fewer iterations may be used if a convergence test deems the process to have completed.
nBins	A single integer giving the number of bins to use for the joint histogram created by the algorithm.
bendingEnergyWeight	A numeric value giving the weight of the bending energy term in the cost function.
linearEnergyWeight	A numeric value giving the weight of the linear energy term in the cost function.
jacobianWeight	A numeric value giving the weight of the Jacobian determinant term in the cost function.
finalSpacing	A numeric vector giving the spacing of control points in the final grid, along the X, Y and Z directions respectively. This is set from the initial control point image, if one is supplied.
spacingUnit	A character string giving the units in which the finalSpacing is specified: either "voxel" for pixels/voxels, or "world" for real-world units (see pixunits).
interpolation	A single integer specifying the type of interpolation to be applied to the final resampled image. May be 0 (nearest neighbour), 1 (trilinear) or 3 (cubic spline). No other values are valid.
verbose	A single logical value: if TRUE, the code will give some feedback on its progress; otherwise, nothing will be output while the algorithm runs. Run time can be seconds or more, depending on the size and dimensionality of the images.
estimateOnly	Logical value: if TRUE, transformations will be estimated, but images will not be resampled.

`sequentialInit` If `TRUE` and `source` has higher dimensionality than `target`, transformations which are not explicitly initialised will begin from the result of the previous registration.

`internal` If `NA`, the default, the final resampled image will be returned as a standard R array, but control point maps will be objects of class `"internalImage"`, containing only basic metadata and a C-level pointer to the full image. (See also `readNifti`.) If `TRUE`, all image-type objects in the result will be internal images; if `FALSE`, they will all be R arrays. The default is fine for most purposes, but using `TRUE` may save memory, while using `FALSE` can be necessary if there is a chance that external pointers will be invalidated, for example when returning from worker threads.

`precision` Working precision for the registration. Using single- precision may be desirable to save memory when coregistering large images.

`threads` For OpenMP-capable builds of the package, the maximum number of threads to use.

Details

This function performs the dual operations of finding a transformation to optimise image alignment, and resampling the source image into the space of the target image (and vice-versa, if `symmetric` is `TRUE`). Unlike `niftyreg.linear`, this transformation is nonlinear, and the degree of deformation may vary across the image.

The nonlinear warping is based on free-form deformations. A lattice of equally-spaced control points is defined over the target image, each of which can be moved to locally modify the mapping to the source image. In order to assess the quality of the warping between the two images, an objective function based on the normalised mutual information is used, with penalty terms based on the bending energy or the squared log of the Jacobian determinant. The objective function value is optimised using a conjugate gradient scheme.

The source image may have 2, 3 or 4 dimensions, and the target 2 or 3. The dimensionality of the target image determines whether 2D or 3D registration is applied, and source images with one more dimension than the target (i.e. 4D to 3D, or 3D to 2D) will be registered volumewise or slicewise, as appropriate. In the latter case the last dimension of the resulting image is taken from the source image, while all other dimensions come from the target. One image of control points is returned for each registration performed.

Value

See `niftyreg`.

Note

Performing a linear registration first, and then initialising the nonlinear transformation with the result (via the `init` parameter), is highly recommended in most circumstances.

Author(s)

Jon Clayden <code@clayden.org>

References

The algorithm used by this function is described in the following publication.

M. Modat, G.R. Ridgway, Z.A. Taylor, M. Lehmann, J. Barnes, D.J. Hawkes, N.C. Fox & S. Ourselin (2010). Fast free-form deformation using graphics processing units. Computer Methods and Programs in Biomedicine 98(3):278-284.

See Also

`niftyreg`, which can be used as an interface to this function, and `niftyreg.linear` for linear registration. Also, `forward` and `reverse` to extract transformations, and `applyTransform` to apply them to new images or points.

`readAffine` *Read an affine matrix from a file*

Description

This function is used to read a 4x4 numeric matrix representing an affine transformation from a file. It is a wrapper around `read.table` which additionally ensures that required attributes are set. The type of the matrix must be specified, as there are differing conventions across software packages.

Usage

```
readAffine(fileName, source = NULL, target = NULL, type = NULL)
```

Arguments

`fileName`	A string giving the file name to read the affine matrix from.
`source`	The source image for the transformation. If `NULL`, the file will be searched for a comment specifying the path to a NIfTI file.
`target`	The target image for the transformation. If `NULL`, the file will be searched for a comment specifying the path to a NIfTI file.
`type`	The type of the affine matrix, which describes what convention is it is stored with. Currently valid values are `"niftyreg"` and `"fsl"` (for FSL FLIRT). If `NULL`, the function will look in the file for a comment specifying the type.

Value

An matrix with class `"affine"`, converted to the NiftyReg convention and with `source` and `target` attributes set appropriately.

Author(s)

Jon Clayden <code@clayden.org>

See Also

read.table, writeAffine

Examples

```
print(readAffine(system.file("extdata","affine.txt",package="RNiftyReg")))
```

---

readNifti *Read a NIfTI-1 format file*

Description

This function reads one or more NIfTI-1 files into R, using the standard NIfTI-1 C library. It extends the equivalent function from the RNifti package with source and target image parameters.

Usage

```
readNifti(file, source = NULL, target = NULL, internal = FALSE)
```

Arguments

file	A character vector of file names.
source, target	If the specified file contains a transformation, these parameters can be used to specify the associated source and target images, which are stored in attributes of the same name. Only used if file is of unit length.
internal	Logical value. If FALSE (the default), an array of class "niftiImage", containing the image pixel or voxel values, will be returned. If TRUE, the return value will be an object of class "internalImage", which contains only minimal metadata about the image. Either way, the return value has an attribute which points to a C data structure containing the full image.

Value

An array or internal image, with class "niftiImage", and possibly also "internalImage".

Author(s)

Jon Clayden <code@clayden.org>

saveTransform *Save and load transform objects*

Description

These objects save a full transformation object, including source and target image metadata, to a self-contained RDS file, or load it back from such a file. This is currently only possible for linear transforms.

Usage

```
saveTransform(transform, file)

loadTransform(file)
```

Arguments

transform	A transform, possibly obtained from forward or reverse.
file	The filename to save to, or load from.

Value

loadTransform returns a deserialised transform object.

Author(s)

Jon Clayden <code@clayden.org>

See Also

writeAffine, readAffine

similarity *Similarity measures between images*

Description

This function calculates a similarity measure between two images, after resampling one into the space of the other. The only supported measure is currently normalised mutual information, which is also used as a cost function by the registration algorithms.

Usage

```
similarity(source, target, targetMask = NULL, interpolation = 3L,
  threads = getOption("RNiftyReg.threads"))
```

Arguments

source	The source image, in any acceptable form.
target	The target image. Must have the same dimensionality as the source image.
targetMask	An optional mask image in target space, whose nonzero region will be the area over which the measure is calculated.
interpolation	A single integer specifying the type of interpolation to be applied to the source image when resampling it into the space of the target image. May be 0 (nearest neighbour), 1 (trilinear) or 3 (cubic spline). No other values are valid.
threads	For OpenMP-capable builds of the package, the maximum number of threads to use.

Value

A single numeric value representing the similarity between the images.

Author(s)

Jon Clayden <code@clayden.org>

See Also

niftyreg

---

translate *Apply simple transformations*

Description

These functions allow simple transformations to be applied quickly, or in a chosen order. They represent simplified interfaces to the buildAffine and applyTransform functions, and are compatible with the chaining operator from the popular magrittr package (although performing one single transformation may be preferable).

Usage

```
translate(source, translation, ...)

rescale(source, scales, anchor = c("none", "origin", "centre", "center"), ...)

skew(source, skews, anchor = c("none", "origin", "centre", "center"), ...)

rotate(source, angles, anchor = c("none", "origin", "centre", "center"), ...)
```

Arguments

source	A 2D or 3D image, in the sense of isImage.
translation	Translations along each axis, in pixunits units. May also be a list, such as that produced by decomposeAffine, with elements for translation, scales, skews and angles.
...	Additional arguments to applyTransform.
scales	Scale factors along each axis.
anchor	The fixed point for the transformation. Setting this parameter to a value other than "none" will override the translation parameter, with the final translation set to ensure that the requested point remains in the same place after transformation.
skews	Skews in the XY, XZ and YZ planes.
angles	Roll, pitch and yaw rotation angles, in radians. If source is two-dimensional, a single angle will be interpreted as being in the plane as expected.

Value

The transformed image.

Author(s)

Jon Clayden <code@clayden.org>

See Also

buildAffine, applyTransform

---

writeAffine *Write an affine matrix to a file*

Description

This function is used to write a 4x4 numeric matrix representing an affine transformation to a file. A comment is also (optionally) written, which specifies the matrix as using the NiftyReg convention, for the benefit of readAffine.

Usage

```
writeAffine(affine, fileName, comments = TRUE)
```

Arguments

affine	A 4x4 affine matrix.
fileName	A string giving the file name to write the matrix to.
comments	Logical value: if TRUE comments are written to the file in lines beginning with #.

Author(s)

Jon Clayden <code@clayden.org>

See Also

`write.table`, `readAffine`

Index