Package 'RNiftyReg'

July 2, 2014

Version 1.1.2

Date 2013-11-07

Title Medical image registration using the NiftyReg library

Author Jon Clayden; based on original code by Marc Modat and Pankaj Daga

Maintainer Jon Clayden <jon.clayden+rniftyreg@gmail.com>

Depends R (>= 2.9.0)

Imports reportr, oro.nifti

Suggests splines

Description This package provides an R interface to the NiftyReg image registration tools <http://sourceforge.net/projects/niftyreg/>.

License GPL-2

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

NeedsCompilation yes

Repository CRAN

Date/Publication 2013-11-07 14:39:47

R topics documented:

convertAffine 2
decomposeAffine 3
getDeformationField 4
invertAffine 5
niftyreg 6
niftyreg.linear 8
niftyreg.nonlinear 10
readAffine 13
transformVoxelToWorld 14

transformWithAffine 15
transformWithControlPoints 16
writeAffine 17
xformToAffine 17

Index **19**

`convertAffine` *Convert an affine matrix between different storage conventions*

Description

Affine transformation matrices can be stored using a number of different conventions. This function converts between conventions.

Usage

```
convertAffine(affine, source, target, newType = c("niftyreg","fsl"),
    currentType = NULL)
```

Arguments

`affine`	A 4x4 matrix representing an affine transformation matrix.
`source`	A `"nifti"` object representing the source image for the transformation.
`target`	A `"nifti"` object representing the target image for the transformation.
`newType`	The convention type to convert to. Currently valid values are `"niftyreg"` and `"fsl"` (for FSL FLIRT).
`currentType`	The current type of the affine matrix, which can be either of the valid values for `newType`. If `NULL`, the code attempts to determine the current type from the `affineType` attribute of the matrix.

Details

The source and target images for the original transformation are required to find the corresponding "xform" matrices, which are needed to perform the conversion.

Value

A copy of the `affine` object provided, converted to the requested storage convention.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

See Also

`readAffine`, `writeAffine`

`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.

Usage

```
decomposeAffine(affine, source = NULL, target = NULL, type = NULL)
```

Arguments

`affine`	A 4x4 matrix representing an affine transformation matrix.
`source`	A `"nifti"` object representing the source image for the transformation. Only required if `type` (or the matrix's `affineType` attribute) is `"niftyreg"`, in which case it is passed to `convertAffine`.
`target`	A `"nifti"` object representing the target image for the transformation. Only required if `type` (or the matrix's `affineType` attribute) is `"niftyreg"`, in which case it is passed to `convertAffine`.
`type`	The storage convention type of the affine matrix, if it is not stored in the `affineType` attribute of the 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 (generally in mm) 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 <jon.clayden+rniftyreg@gmail.com>

See Also

`convertAffine`, `niftyreg`

`getDeformationField` *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.

Usage

```
getDeformationField(target, affine = NULL, controlPointImage = NULL,
    jacobian = TRUE)
```

Arguments

`target`	A `"nifti"` object representing the target image for the transformation.
`affine`	A 4x4 matrix representing an affine matrix, using the `"niftyreg"` convention. Either this or a control point image must be specified.
`controlPointImage`	A `"nifti"` object representing the control points for a nonlinear transformation. Either this or an affine matrix must be specified.
`jacobian`	A logical value: if `TRUE`, a Jacobian determinant map is also calculated and returned.

Value

A list with components

`deformationField`
: An object of class `"nifti"` representing the deformation field.

`jacobian`
: An object of class `"nifti"` representing the Jacobian determinant map, if requested. Otherwise this element is not present.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

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 preserves the `affineType` attribute.

Usage

```
invertAffine(affine)
```

Arguments

`affine`
: An existing 4x4 affine matrix.

Value

The inverted affine matrix.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

See Also

`solve`

Examples

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

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, targetMask = NULL, initAffine = NULL,
    scope = c("affine","rigid","nonlinear"), estimateOnly = FALSE, ...)
```

Arguments

source	The source image, an object of class "nifti" with 2, 3 or 4 dimensions. Package oro.nifti defines this class and provides functions for reading and writing NIfTI files.
target	The target image, an object of class "nifti" with 2 or 3 dimensions.
targetMask	An optional mask image (again a "nifti" object), whose nonzero region will be taken as the region of interest for the registration. Must have the same voxel and image dimensions as the target image.
initAffine	An optional affine matrix, or list of matrices, to initialise the algorithm.
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).
estimateOnly	A single logical value: if TRUE, transformations will be estimated but images will not be resampled.
...	Further arguments to niftyreg.linear or niftyreg.nonlinear.

Value

A list of class "niftyreg" with components

image	An image object of class "nifti" representing the registered and resampled source image in the space of the target image. This element is NULL if the estimateOnly parameter is TRUE.
affine	A list of 4x4 matrices containing the optimised affine transformations for each slice or volume of the source image. If the target is a 2D image, elements involving the Z dimension will have no effect. This element is NULL if nonlinear registration is performed.
control	A list of objects of class "nifti", representing the control point images for each warping. This element is NULL if linear (rigid or affine) registration is performed.

reverseImage	An image object of class "nifti" representing the reverse-registered target image in the space of the source image. This element is NULL unless symmetric nonlinear registration is performed and the estimateOnly parameter is FALSE.
reverseControl	A list of objects of class "nifti", representing the control points for each reverse warping. This element is NULL unless symmetric nonlinear registration is performed.
iterations	A list of integer vectors giving the number of iterations actually run within each level, for each slice or volume of the source image. Note that for the first level of the linear algorithm specifically, twice the specified number of iterations is allowed.
scope	Copied from the function argument of the same name.

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, target)`.

There is no reason that arrays that do not represent medical images cannot be registered using this function. A standard R array can be converted to a valid "nifti" object easily for these purposes using the as.nifti function in the oro.nifti package.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

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. See nifti (no relation!), in the oro.nifti package, for creating the image objects passed to this function. Useful related functions are as.nifti, readNIfTI and writeNIfTI.

Examples

```
## Not run:
source <- oro.nifti::readNIfTI(system.file("extdata","source.nii.gz",
  package="RNiftyReg"))
target <- oro.nifti::readNIfTI(system.file("extdata","target.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. A precalculated transformation can be applied to a new image using the `applyAffine` function.

Usage

```
niftyreg.linear(source, target, targetMask = NULL, initAffine = NULL,
    scope = c("affine","rigid"), nLevels = 3, maxIterations = 5,
    useBlockPercentage = 50, finalInterpolation = 3, verbose = FALSE,
    estimateOnly = FALSE)

applyAffine(affine, source, target, affineType = NULL,
    finalInterpolation = 3)
```

Arguments

`source`	The source image, an object of class `"nifti"` with 2, 3 or 4 dimensions. Package `oro.nifti` defines this class and provides functions for reading and writing NIfTI files.
`target`	The target image, an object of class `"nifti"` with 2 or 3 dimensions.
`targetMask`	An optional mask image (again a `"nifti"` object), whose nonzero region will be taken as the region of interest for the registration. Must have the same voxel and image dimensions as the target image.
`initAffine`	An optional affine matrix, or list of matrices, to initialise the algorithm. If `NULL`, the identity matrix is used, with an appropriate offset to account for differences in the image origins.
`scope`	A string describing the scope, or number of degrees of freedom (DOF), of the registration. Only `"affine"` (12 DOF) and `"rigid"` (6 DOF) are currently supported.
`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.

`finalInterpolation`
: 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.

`estimateOnly`
: A single logical value: if `TRUE`, the transformation will be estimated but the source image will not be resampled.

`affine`
: For `applyAffine`, the affine transformation(s) to apply to the source image.

`affineType`
: For `applyAffine`, the storage convention type of the affine matrix, if it is not stored in the `affineType` attribute of the matrix.

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.

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.

The `applyAffine` function is a convenience wrapper that calls `niftyreg.linear` with `nLevels=0` to apply the specified transformation without any further optimisation. Note that a target image must still be specified in this case, since the metadata associated with that image is needed by `niftyreg.linear`.

Value

See `niftyreg`.

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.linear(source, target, target)`.

There is no reason that arrays that do not represent medical images cannot be registered using this function. A standard R array can be converted to a valid `"nifti"` object easily for these purposes using the `as.nifti` function in the `oro.nifti` package.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

References

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

S. Ourselin, A. Roche, G. Subsol, X. Pennec & N. Ayache (2000). Reconstructing a 3D structure from serial histological sections. Image and Vision Computing 19(1-2):25-31.

S. Ourselin, R. Stefanescu & X. Pennec (2002). Robust registration of multi-modal images: towards real-time clinical applications. Medical Image Computing and Computer-Assisted Intervention. Vol. 2489 of Lecture Notes in Computer Science, pp. 140-147.

See Also

`niftyreg`, which can be used as an interface to this function, and `niftyreg.nonlinear` for nonlinear registration. Also, `transformWithAffine` for transforming points, rather than images, using the estimated affine matrices. See `nifti` (no relation!), in the `oro.nifti` package, for creating the image objects passed to this function. Useful related functions are `as.nifti`, `readNIfTI` and `writeNIfTI`.

`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. A precalculated transformation can be applied to a new image using the `applyControlPoints` function.

Usage

```
niftyreg.nonlinear(source, target, targetMask = NULL, initAffine = NULL,
    initControl = NULL, symmetric = FALSE, sourceMask = NULL, nLevels = 3,
    maxIterations = 300, nBins = 64, bendingEnergyWeight = 0.005,
    jacobianWeight = 0, inverseConsistencyWeight = 0.01,
    finalSpacing = c(5,5,5), spacingUnit = c("vox","mm"),
    finalInterpolation = 3, verbose = FALSE, estimateOnly = FALSE)

applyControlPoints(controlPointImage, source, target,
    finalInterpolation = 3)
```

Arguments

source	The source image, an object of class "nifti" with 2, 3 or 4 dimensions. Package oro.nifti defines this class and provides functions for reading and writing NIfTI files.
target	The target image, an object of class "nifti" with 2 or 3 dimensions.
targetMask	An optional mask image (again a "nifti" object), whose nonzero region will be taken as the region of interest for the registration. Must have the same voxel and image dimensions as the target image.
initAffine	An optional affine matrix, or list of matrices, to initialise the algorithm. If both this parameter and initControl are NULL, the identity matrix is used, with an appropriate offset to account for differences in the image origins.
initControl	An optional image of class "nifti", or a list of images, representing fields of previously-calculated control points to use as initialisation for the algorithm. This parameter takes priority over initAffine if both are not NULL, but may not be specified if symmetric is TRUE.
symmetric	A single logical value: if TRUE, a symmetric version of the registration algorithm is used to simultaneously obtain transformations from source to target and target to source. There are some restrictions in this mode: see Details.
sourceMask	An optional mask image whose nonzero region will be taken as the region of interest for the registration. Must have the same voxel and image dimensions as the source image. Ignored if symmetric is FALSE.
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.
jacobianWeight	A numeric value giving the weight of the Jacobian determinant term in the cost function.
inverseConsistencyWeight	A numeric value giving the weight of the term ensuring inverse consistency in the cost function. Ignored if symmetric is FALSE.
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 initControl image if one is supplied.
spacingUnit	A character string giving the units in which the finalSpacing is specified: either "vox" for voxels or "mm" for millimetres (which is assumed to be the spatial unit of the source and target images).

`finalInterpolation`
: 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.

`estimateOnly`
: A single logical value: if `TRUE`, the transformation(s) will be estimated but the image(s) will not be resampled.

`controlPointImage`
: For `applyControlPoints`, the control point map to apply to the source image.

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.

The `symmetric` option allows registration to be performed in both directions simultaneously. This may be preferable to performing two one-way registrations, since a constraint is applied to ensure that the control point maps in the two directions are consistent. However, the source and target images must have the same dimensionality in this case, and an initial control point map may not be used.

The `applyControlPoints` function is a convenience wrapper that calls `niftyreg.nonlinear` with `nLevels=0` to apply the specified transformation without any further optimisation. Note that a target image must still be specified in this case, since the metadata associated with that image is needed by `niftyreg.nonlinear`.

Value

See `niftyreg`.

Note

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

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.nonlinear(source, target, target)`.

There is no reason that arrays that do not represent medical images cannot be registered using this function. A standard R array can be converted to a valid `"nifti"` object easily for these purposes using the `as.nifti` function in the `oro.nifti` package.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

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, `transformWithControlPoints` for transforming points, rather than images, using the estimated control points. See `nifti` (no relation!), in the `oro.nifti` package, for creating the image objects passed to this function. Useful related functions are `as.nifti`, `readNIfTI` and `writeNIfTI`.

`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 the `affineType` attribute is set. This is required because there are different conventions for storing affine matrices.

Usage

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

Arguments

`fileName`	A string giving the file name to read the affine matrix from.
`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 affine matrix with `affineType` attribute set appropriately.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

See Also

`read.table`, `writeAffine`, `convertAffine`

Examples

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

---

transformVoxelToWorld *Transform points between voxel and "world" coordinates*

Description

These functions are used to transform points from dimensionless pixel or voxel coordinates to "real-world" coordinates, typically in millimetres, and back.

Usage

```
transformVoxelToWorld(points, image, simple = FALSE, ...)

transformWorldToVoxel(points, image, simple = FALSE, ...)
```

Arguments

`points`	A vector giving the coordinates of a point, or a matrix with one point per row.
`image`	The image in whose space the points are given, an object of class `"nifti"`.
`simple`	A logical value: if `TRUE` then the transformation is performed simply by rescaling the points according to the voxel dimensions recorded in the `image`. Otherwise the full xform matrix is used.
`...`	Additional arguments to `xformToAffine`.

Value

A vector or matrix of transformed points.

Note

Voxel coordinates are assumed by these functions to use R's indexing convention, beginning from 1.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

See Also

xformToAffine

---

transformWithAffine *Transform points using an affine matrix*

Description

This function is used to transform points from source to target space using an affine transformation, possibly obtained from niftyreg.linear. The exact subvoxel location in target space is returned.

Usage

```
transformWithAffine(points, affine, source, target, type = NULL)
```

Arguments

points	A vector giving the coordinates of a point, or a matrix with one point per row. These must be given as voxel-based locations: see transformWorldToVoxel for a function to convert from "real world" coordinates.
affine	A 4x4 matrix representing an affine transformation matrix.
source	A "nifti" object representing the source image for the transformation.
target	A "nifti" object representing the target image for the transformation.
type	The convention type of the affine matrix. Currently valid values are "niftyreg" and "fsl" (for FSL FLIRT). If NULL, the code attempts to determine the current type from the affineType attribute of the matrix.

Value

A vector or matrix of transformed points.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

See Also

For transforming points using the xform matrix stored in an image, transformVoxelToWorld provides a simpler interface.

`transformWithControlPoints` *Transform points using a control point image*

Description

This function is used to transform points from source to target space using a control point image, possibly obtained from `niftyreg.nonlinear`. A subvoxel location is target space is returned unless the nearest voxel is requested.

Usage

```
transformWithControlPoints(points, controlPointImage, source, target,
    nearest = FALSE)
```

Arguments

`points`	A vector giving the coordinates of a point, or a matrix with one point per row. These must be given as voxel-based locations: see `transformWorldToVoxel` for a function to convert from "real world" coordinates.
`controlPointImage`	An object of class `"nifti"` containing the control points parameterising the transformation.
`source`	A `"nifti"` object representing the source image for the transformation.
`target`	A `"nifti"` object representing the target image for the transformation.
`nearest`	A logical value: if `TRUE`, the return value contains just the location of the nearest voxel to each requested location. This is computationally more efficient than simply rounding the result.

Details

The transformation performed by this function is not exact, and 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. `getDeformationField`), 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 cubic spline regression.

Value

A vector or matrix of transformed points.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

See Also

niftyreg.nonlinear, getDeformationField

---

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. The affineType attribute is also written, as a comment.

Usage

```
writeAffine(affine, fileName)
```

Arguments

affine	A 4x4 affine matrix.
fileName	A string giving the file name to write the matrix to.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

See Also

write.table, readAffine, convertAffine

---

xformToAffine *Create an affine matrix corresponding to the "xform" of an image*

Description

This function converts the "qform" or "sform" information in a NIfTI header into its corresponding affine matrix. These two "xform" mechanisms are defined by the NIfTI standard and may both be in use in a particular image header.

Usage

```
xformToAffine(image, useQuaternionFirst = TRUE)
```

Arguments

image	A object of class "nifti" from which to take the information.
useQuaternionFirst	A single logical value. If TRUE, the "qform" matrix will be used first, if it is defined; otherwise the "sform" matrix will take priority.

Value

A affine matrix corresponding to the "qform" or "sform" information in the image header.

Author(s)

Jon Clayden <jon.clayden+rniftyreg@gmail.com>

References

The NIfTI-1 standard (http://nifti.nimh.nih.gov/nifti-1) is the definitive reference on "xform" conventions.

See Also

nifti, in the oro.nifti package

Index

applyAffine *(*niftyreg.linear*)*, 8
applyControlPoints
*(*niftyreg.nonlinear*)*, 10

convertAffine, 2, *3, 4*, *14*, *17*

decomposeAffine, 3

getDeformationField, 4, *16, 17*

invertAffine, 5

niftyreg, *4*, 6, *9, 10*, *12, 13*
niftyreg.linear, *5–7*, 8, *12, 13*, *15*
niftyreg.nonlinear, *5–7*, *10*, 10, *16, 17*

read.table, *14*
readAffine, *2*, 13, *17*

solve, *5*

transformVoxelToWorld, 14, *15*
transformWithAffine, *10*, 15
transformWithControlPoints, *13*, 16
transformWorldToVoxel, *15, 16*
transformWorldToVoxel
*(*transformVoxelToWorld*)*, 14

write.table, *17*
writeAffine, *2*, *14*, 17

xformToAffine, *14, 15*, 17