Package 'nat.templatebrains'

October 13, 2022

Type Package

Title NeuroAnatomy Toolbox ('nat') Extension for Handling Template Brains

Version 1.0

Description Extends package 'nat' (NeuroAnatomy Toolbox) by providing objects and functions for handling template brains.

License GPL-3

URL http://natverse.org/nat.templatebrains/, https://github.com/natverse/nat.templatebrains

Depends nat (>= 1.8.6), R (>= 2.10), rgl

Imports digest, igraph, memoise, rappdirs

Suggests spelling, git2r (>= 0.22.1), nabor, testthat (>= 2.1.0), covr

Encoding UTF-8

RoxygenNote 7.1.1

BugReports https://github.com/natverse/nat.templatebrains/issues

Language en-GB

NeedsCompilation no

Author Gregory Jefferis [aut, cre] (<https://orcid.org/0000-0002-0587-9355>), James Manton [aut] (<https://orcid.org/0000-0001-9260-3156>)

Maintainer Gregory Jefferis <jefferis@gmail.com>

Repository CRAN

Date/Publication 2020-10-19 14:20:02 UTC

R topics documented:

`add_reglist` *Add reglist object describing a bridging/mirroring registration*

Description

By specifying either `reference`, `sample` *or* `mirror` arguments, you can add a bridging or mirroring registration, respectively, to the list of those in use for `xform_brain` and `mirror_brain`.

Usage

```
add_reglist(
  x,
  reference = NULL,
  sample = NULL,
  mirror = NULL,
  temp = TRUE,
  ...
)
```

Arguments

`x`	A single `reglist` object (which)
`reference, sample`	The reference and sample brains (in `character` or `templatebrain` form) for a bridging registration.
`mirror`	The reference brain (in `character` or `templatebrain` form) for a mirroring registration.
`temp`	Whether to store the on disk representation in a session-specific temporary folder (that will be removed when R closes). Defaults to `TRUE`.
`...`	Additional arguments passed to `saveRDS` e.g. to control compression when the reglist object is saved to disk.

See Also

add_reg_folders

Examples

```
## Not run:
library(nat.flybrains)
# mirroring registration for a specific template brain object
add_reglist(mirroring, mirror=JFRC2013)
# equivalent but withhout needing to construct the template
add_reglist(mirroring, mirror="JFRC2013")

# add a bridging registration between two brains
add_reglist(bridging, reference=JFRC2, sample=JFRC2013)


## End(Not run)
```

`add_reg_folders` *Set or list local folders containing registrations for nat.templatebrains*

Description

`add_reg_folders` sets options('nat.templatebrains.regdirs') appropriately so that registrations can be found by e.g. `xform_brain`.

`extra_reg_folders` lists extra registration folders present in standard location

Usage

```
add_reg_folders(dir = extra_reg_folders(), first = TRUE)

extra_reg_folders(full.names = TRUE)
```

Arguments

`dir`	Path to one or more folders containing registrations. Default value will scan for registration folders in a standard location. (Please see **Details** and **File layout** sections)
`first`	Whether the new folder should be added to the start (default) or end of the search list.
`full.names`	Whether to list full path to registration folders

Details

When `dir` is unset then it will default to the value of `extra_reg_folders()` i.e. any folders / cloned repositories in the standard location

File layout

You must pass a folder containing one or more registrations, not the registration folder itself. So if you have this situation on disk

- myregistrations/
- myregistrations/reg1.list
- myregistrations/reg2.list

you should write `add_reg_folders("/path/to/myregistrations")`

Examples

```
## Not run:
  add_reg_folders("myextraregistrations")

## End(Not run)
# adding a non-existent folder will generate an error
tools::assertError(add_reg_folders(tempfile()))
```

`allreg_dataframe` *Make data.frame with details of all registrations*

Description

Make data.frame with details of all registrations

Usage

```
allreg_dataframe(regdirs = getOption("nat.templatebrains.regdirs"))
```

Arguments

`regdirs`	Character vector of directories to search for registrations (see details)

Details

by default `regdirs` is set to getOption('nat.templatebrains.regdirs')

Value

data.frame with one row for each observed registration and columns

- path
- name
- dup
- bridge

- reference
- sample

If there are no registrations, there will be a data.frame with 0 rows and these columns.

Examples

```
## Not run:
allreg_dataframe()

## End(Not run)
```

`all_templatebrains` *Find all template brains or those matching a given image volume*

Description

`all_templatebrains` returns a data.frame detailing all `templatebrain` objects on the search path (including those inside packages).

Usage

```
all_templatebrains(cached = TRUE, remove.duplicates = FALSE)

guess_templatebrain(
  x,
  rval = c("templatebrain", "name"),
  cached = TRUE,
  mustWork = FALSE
)
```

Arguments

`cached`	When `TRUE` returns precomputed (memoised) results, otherwise rescans searching for all template brains.
`remove.duplicates`	Whether to remove duplicate template brains (as determined by md5 hash) from the result list
`x`	A `im3d` image object, array or matrix compatible with `as.templatebrain` OR a 2 or 3-vector defining the dimensions of an image or image stack.
`rval`	Whether to return the `templatebrain` object itself or just its name.
`mustWork`	Whether to insist that exactly one template brain is found

Value

For `all_templatebrains`, a `data.frame` containing the following columns:

- object The name of the `templatebrain` object
- pos An integer specifying the environment
- package Character vector naming the environment
- md5 md5 hash of the `templatebrain` object
- name
- W,H,D Width, height and depth of image stack (pixels)

`guess_templatebrain` returns a `templatebrain` object when `rval='templatebrain'` or a character vector when `rval='name'`.

See Also

`templatebrain`

Examples

```
## Not run:
all_templatebrains()

guess_templatebrain(im3d(dims=c(30,40,50)))
# or
guess_templatebrain(c(30,40,50))
guess_templatebrain('path/to/my/image.nrrd')

if(require('nat.flybrains')){
  guess_templatebrain(im3d(dims=c(1024,512,218)), rval = 'name')
  # get the matching template brain
  tb=guess_templatebrain(im3d(dims=c(1024,512,218)))
  # get its voxel dimensions
  voxdims(tb)

  tb=guess_templatebrain(c(1024,512))
  tb
}

## End(Not run)
```

---

as.templatebrain *Use image file or other object to initialise template brain*

Description

Use image file or other object to initialise template brain

Usage

```
as.templatebrain(x, ...)

## S3 method for class 'character'
as.templatebrain(x, ...)

## S3 method for class 'im3d'
as.templatebrain(x, regName = NULL, name = regName, ...)

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

Arguments

x	object used to construct the templatebrain, either a character vector with the path to a file or an im3d object.
...	additional named arguments passed to methods and then on to templatebrain that will be added as fields to the templatebrain object.
name, regName	name and short name of the template brain. Will use the filename (minus final extension) by default for both fields.

Details

as.templatebrain can extract the key fields defining an template space from an image file. This is generally a much more convenient approach to defining a templatebrain object than specifying all fields by hand.

Value

A list with class templatebrain

See Also

templatebrain, im3d

Examples

```
# Make templatebrain object using image info from the template brain NRRD file
nhdr=system.file('images','FCWB.nhdr', package='nat.templatebrains')
as.templatebrain(nhdr, name = "FlyCircuit Whole Brain")
```

bridging_graph *Make or query connected graph of bridging registrations*

Description

These functions are designed for expert use. In general it is recommended to use `xform_brain`.

`bridging_graph` creates an igraph::graph representing all known template brains (vertices) and the bridging registrations connecting them (edges).

`shortest_bridging_seq` finds the shortest bridging sequence on a graph of all available bridging registrations, subject to constraints defined by graph connectivity and the `reciprocal parameter`.

Usage

```
bridging_graph(
  regdirs = getOption("nat.templatebrains.regdirs"),
  reciprocal = NA
)

shortest_bridging_seq(
  sample,
  reference,
  via = NULL,
  checkboth = TRUE,
  imagedata = FALSE,
  reciprocal = NA,
  ...
)
```

Arguments

`regdirs`	Character vector of directories to search for registrations (see details)
`reciprocal`	Sets the weight of reciprocal edges in the graph (and thereby whether inverse registrations will be considered).
`sample`	Source template brain (e.g. IS2) that data is currently in. Specified either as character vector or a `templatebrain` object.
`reference`	Target template brain (e.g. IS2) that data should be transformed into.
`via`	(optional) intermediate template brain that the registration sequence must pass through.
`checkboth`	When `TRUE` will look for registrations in both directions. See details.
`imagedata`	Whether `x` should be treated as image data (presently only supported as a file on disk) or 3D object vertices - see details.
`...`	additional arguments passed on to `bridging_graph`

Details

When `reciprocal != NA` we create a graph where each forward transformation is matched by a corresponding inverse transformation with the specified edge weight. The edge weight for forward transforms will always be 1.0.

By default `regdirs` is set to getOption('nat.templatebrains.regdirs')

See Also

`allreg_dataframe`, `xform_brain`

Examples

```
## Not run:
plot(bridging_graph(), vertex.size=25, edge.arrow.size=0.5)
# with reciprocal edges
plot(bridging_graph(reciprocal=3), vertex.size=25)

## End(Not run)
## Not run:
shortest_bridging_seq(FCWB, IS2)
# or
shortest_bridging_seq('FCWB', 'IS2')

shortest_bridging_seq(sample='FCWB', reference='IS2', via="JFRC2")

## End(Not run)
```

---

bridging_sequence *Find sequence of one or more bridging registrations*

Description

This function is primarily intended for developer use (it is used inside `xform_brain`) but may be useful for end users.

Usage

```
bridging_sequence(
  sample,
  reference,
  via = NULL,
  imagedata = FALSE,
  checkboth = !imagedata,
  mustWork = FALSE
)
```

Arguments

sample	Source template brain (e.g. IS2) that data is currently in. Specified either as character vector or a templatebrain object.
reference	Target template brain (e.g. IS2) that data should be transformed into.
via	(optional) intermediate template brain that the registration sequence must pass through.
imagedata	Whether x should be treated as image data (presently only supported as a file on disk) or 3D object vertices - see details.
checkboth	whether to look for registrations in both directions. The default (checkboth=FALSE) will only return registrations in the forward direction (see details).
mustWork	whether to error out if appropriate registrations are not found.

Details

When checkboth=FALSE, only registrations that can be directly used to map image data from sample to reference are returned. When working with 3D points, use checkboth=TRUE. Note that all possible directories will first be scanned for registrations in the preferred direction and then rescanned for the opposite direction if nothing is found.

Registration direction

When mapping points from JFRC2 -> IS2 -> FCWB (i.e. sample=JFRC2, via=IS2, ref=FCWB) the command line passed to CMTK's streamxform should look like: `streamxform -- JFRC2_IS2.list --inverse FCWB_IS2.l`
However when mapping image data the command line for CMTK's reformatx should look like:
`reformatx -o out.nrrd --floating JFRC2.nrrd FCWB.nrrd FCWB_IS2.list --inverse JFRC2_IS2.list`
bridging_sequence produces output like

```
list(JFRC2 = structure(
     "/GD/dev/R/nat.flybrains/inst/extdata/bridgingregistrations/JFRC2_IS2.list",
       swap = TRUE),
   IS2 = "/GD/dev/R/nat.flybrains/inst/extdata/bridgingregistrations/FCWB_IS2.list")
```

in these circumstances, which xformpoints.cmtkreg turns into "– JFRC2_IS2.list –inverse FCWB_IS2.list".

Examples

```
## Not run:
bridging_sequence(sample=JFRC2, ref=FCWB, checkboth = T)
bridging_sequence(sample=JFRC2, via=IS2, ref=FCWB, checkboth = T)

## End(Not run)
```

display_slice *Display an image slice in 3D*

Description

Display an image slice in 3D

Usage

```
display_slice(brain, slice, ...)
```

Arguments

brain	template brain (e.g. IS2) of the slice.
slice	Path to PNG image containing slice to display.
...	extra arguments to pass to persp3d.

download_reg_repo *Download and register git repository containing registrations*

Description

Note that these extra registrations will be downloaded to a standard location on your hard drive that will be used for one session to the next. See examples and local_reg_dir_for_url.

Usage

```
download_reg_repo(url, localdir = NULL, ...)
```

Arguments

url	Location of one or more remote git repositories. Can accept partial github specifications of the form "<user>/<repo>".
localdir	Full path to local checkout location of git repository. When localdir=NULL, the default, a sensible location is chosen using the rappdirs function.
...	additional arguments passed to git2r::clone e.g. credentials for private repo.

See Also

add_reg_folders, local_reg_dir_for_url, git2r::clone

update_reg_repos

Examples

```
## find the root location of all registration directories
local_reg_dir_for_url()
## Not run:
## Add the two main jefferislab bridging and mirroring registration
# collections for Drosophila brains from github.com.
download_reg_repo("jefferislab/BridgingRegistrations")
download_reg_repo("jefferislab/MirrorRegistrations")

## update all current registration repositories
update_reg_repos()

## End(Not run)
```

FCWB.demo *Sample template brain: FlyCircuit Whole Brain*

Description

This is a sample template brain for testing purposes which is equivalent to the FCWB template brain defined by the `nat.flybrains`, which should be considered the canonical version.

---

local_reg_dir_for_url *Standard local checkout location for extra registration directories*

Description

Standard local checkout location for extra registration directories

Usage

```
local_reg_dir_for_url(url = NULL)
```

Arguments

`url`	Character vector containing a url. When `url=NULL` defaults to giving the base path.

Details

When called without any argument returns the root directory that will be inspected for extra registrations. You can put a sub-folder yourself there manually and then call add_reg_folders, but you are much better off in general using `download_reg_repo` to install from a github repository such as this one of ours:. jefferislab/BridgingRegistrations

Note that this folder will always be the same place on a machine i.e. this defines a consistent, persistent location on disk to store data across sessions.

When called with a url, a SHA1 hash will be calculated for the URL and appended to the basepath. This should ensure that locations derived from different URLs do not clash.

See Also

download_reg_repo

---

mirror_brain *Mirror 3D object around a given axis, optionally using a warping registration*

Description

Mirror 3D object around a given axis, optionally using a warping registration

Usage

```
mirror_brain(
  x,
  brain = regtemplate(x),
  mirrorAxis = c("X", "Y", "Z"),
  transform = c("warp", "affine", "flip"),
  ...
)
```

Arguments

x	the 3D object to be mirrored.
brain	source template brain (e.g. IS2) that data is in.
mirrorAxis	the axis to mirror (default "X").
transform	whether to use warp (default) or affine component of registration, or simply flip about midplane of axis.
...	extra arguments to pass to mirror.

See Also

xform_brain, regtemplate

Examples

```
data(FCWB.demo)
# Simple mirror along the x i.e. medio-lateral axis
kcs20.flip=mirror_brain(kcs20, FCWB.demo, transform='flip')

## Full non-rigid mirroring to account for differences in shape/centering of
## template brain.
## Depends on nat.flybrains package and system CMTK installation
## Not run:
library(nat.flybrains)
kcs20.right=mirror_brain(kcs20, FCWB, .progress='text')
plot3d(kcs20, col='red')
```

```
plot3d(kcs20.right, col='green')
# include surface plot of brain
plot3d(FCWB)

# Compare simple flip with full mirror
# This template brain is highly symmetric so these are almost identical
clear3d()
plot3d(kcs20.flip, col='blue')
plot3d(kcs20.right, col='green')

# Convert to JFRC2 and do the same
kcs20.jfrc2=xform_brain(kcs20, sample = FCWB, reference=JFRC2, .progress='text')
kcs20.jfrc2.right=mirror_brain(kcs20.jfrc2, JFRC2, .progress='text')
kcs20.jfrc2.flip=mirror_brain(kcs20.jfrc2, JFRC2, transform='flip')
clear3d()
# This time there is a bigger difference between the two transformations
plot3d(kcs20.jfrc2.flip, col='blue')
plot3d(kcs20.jfrc2.right, col='green')
# plot mushroom body neuropils as well
plot3d(JFRC2NP.surf, "MB.*_R", alpha=0.3, col='grey')

# Compare Euclidean distance between corresponding points in all neurons
diffs=xyzmatrix(kcs20.jfrc2.flip)-xyzmatrix(kcs20.jfrc2.right)
hist(sqrt(rowSums(diffs^2)), xlab='Distance /microns')

## End(Not run)
```

`plot3d.templatebrain` *Plot 3D surface of a template brain*

Description

Plot 3D surface of a template brain

Usage

```
## S3 method for class 'templatebrain'
plot3d(x, col = "grey", alpha = 0.3, ...)
```

Arguments

`x`	the template brain to plot.
`col`	the color of the surface.
`alpha`	the alpha value of the surface.
`...`	extra arguments to pass to `plot3d`.

Details

This function will work immediately for the standard templatebrain defined in the package documentation. If passed an object called e.g. FCWB it expects to find another object named FCWB.surf containing the surface information. If you follow this naming convention for user-defined refbrains it will work for them as well.

`regtemplate` *Get or set the registration template space in which an object lives*

Description

Get or set the registration template space in which an object lives

Usage

```
regtemplate(x)

regtemplate(x) <- value
```

Arguments

`x`	The 3D object whose registration space will be set/returned
`value`	The registration template brain (either a character vector naming the space or a `templatebrain` object)

Details

In order to facilitate transformations between objects in defined anatomical spaces these functions allow the registration template for an object to be specified. Most of the time you will not need to use these functions manually since the appropriate space will be set by the function xform_brain and friends.

Value

Either a `templatebrain` object or the newly tagged object

Examples

```
## Not run:
library(nat.flybrains)
kcs3=kcs20[1:3]
regtemplate(kcs3)=FCWB
regtemplate(kcs3)

kcs3m=mirror_brain(kcs3, brain=regtemplate(kcs20))
plot3d(kcs3, col='red')
plot3d(kcs3m, col='green')

## End(Not run)
```

templatebrain *Construct templatebrain object for an image registration template*

Description

`templatebrain` objects encapsulate key information for the reference brain in an image registration. Usually this will be a standard template brain used for many registrations. **It will normally be much more convenient to use** `as.templatebrain` **methods to convert an image file or an im3d object into a** `templatebrain`.

Usage

```
templatebrain(
  name,
  regName = name,
  type = NULL,
  sex = NULL,
  dims = NULL,
  BoundingBox = NULL,
  voxdims = NULL,
  origin = NULL,
  units = NULL,
  description = NULL,
  doi = NULL,
  ...
)
```

Arguments

`name`	the full name of the template.
`regName`	the short name. This will be the stem used to prefix registrations (e.g. JFRC2_someimage.list) for this template brain and likely also the stem of the template brain image (e.g. JFRC2.nrrd).
`type`	one of `c('single brain', 'average')`, indicating whether the template brain has been created from just one image, or is the average of multiple images.
`sex`	the sex of the template brain. For templates with `type=='average'`, the possibility of `sex='intersex'` exists.
`dims`	dimensions of the image (number of voxels).
`BoundingBox`	physical dimensions of the image (see `boundingbox`).
`voxdims`	physical spacing between voxels.
`origin`	the physical location of the first voxel
`units`	units of physical measurements (e.g. microns).
`description`	details of the template.
`doi`	a DOI for the original template brain image.
`...`	additional named arguments that will be added as fields to the `templatebrain` object.

Details

A variety of methods are available to work on templatebrain objects. See templatebrain-meths for basic methods. The two main functions that are available for using template brains are xform_brain and mirror_brain.

templatebrain objects are only useful for transformation processes when the BoundingBox is specified to define the physical extent of the volume. We use the definition of the Amira 3D visualisation and analysis software. This corresponds to the **node** centers option in the NRRD format. The bounding box can be obtained from NRRD or AmiraMesh format files. See boundingbox for details.

Value

A list with class templatebrain.

See Also

as.templatebrain, templatebrain-meths, xform_brain, mirror_brain.

---

templatebrain-meths *Template brain methods*

Description

is.templatebrain tests if object is of class templatebrain

as.character.templatebrain converts template brain to character vector representation (normally used to extract the short name i.e. regName).

print.templatebrain prints templatebrain information in human-readable form

as.im3d converts a template brain to a nat::im3d object; this is probably useful for developers.

origin extracts the space origin of a templatebrain object.

dim extracts the dimensions (in number of pixels) of the image associated with a templatebrain object.

voxdims extracts the dimensions (in calibrated spatial units, e.g. microns) of voxels in the image associated with a templatebrain object.

boundingbox extracts the boundingbox (in calibrated spatial units, e.g. microns) of the image associated with a templatebrain object. See boundingbox for details.

Usage

```
is.templatebrain(x)

## S3 method for class 'templatebrain'
as.character(x, field = c("regName", "name"), ...)

## S3 method for class 'templatebrain'
```

```
print(x, ...)

## S3 method for class 'templatebrain'
as.im3d(x, ...)

## S3 method for class 'templatebrain'
origin(x, ...)

## S3 method for class 'templatebrain'
dim(x, ...)

## S3 method for class 'templatebrain'
voxdims(x, ...)

## S3 method for class 'templatebrain'
boundingbox(x, ...)
```

Arguments

x	an object (usually a templatebrain).
field	which field to use (defaults to 'regName').
...	additional arguments for methods.

Value

A logical indicating whether or not the object is a templatebrain.

Character vector.

See Also

im3d

origin

voxdims

boundingbox

Examples

```
data(FCWB.demo)
is.templatebrain(FCWB.demo)
origin(FCWB.demo)
dim(FCWB.demo)
voxdims(FCWB.demo)
boundingbox(FCWB.demo)
# print method
FCWB.demo
```

update_reg_repos *Update local copy of git repository containing registrations*

Description

When x=NULL all repositories listed in options(nat.templatebrains.regdirs) are checked to see if they are git repositories and, if yes, they are pulled to update.

Usage

```
update_reg_repos(x = NULL)
```

Arguments

x	Path to local checkout of a registration git repository. See details for meaning of default.

See Also

download_reg_repo

xform_brain *Transform 3D object between template brains*

Description

Transform 3D object between template brains

Usage

```
xform_brain(
  x,
  sample = regtemplate(x),
  reference,
  via = NULL,
  imagedata = is.character(x),
  checkboth = NULL,
  target = NULL,
  Verbose = interactive(),
  ...
)
```

Arguments

x	the 3D object to be transformed
sample	Source template brain (e.g. IS2) that data is currently in. Specified either as character vector or a templatebrain object.
reference	Target template brain (e.g. IS2) that data should be transformed into.
via	(optional) intermediate template brain that the registration sequence must pass through.
imagedata	Whether x should be treated as image data (presently only supported as a file on disk) or 3D object vertices - see details.
checkboth	When TRUE will look for registrations in both directions. See details.
target	When transforming image data, this specifies the target space (defaults to reference when imagedata=TRUE). See Details.
Verbose	Whether to show a message with the sequence of template brains
...	extra arguments to pass to xform and then on to xformpoints or xformimage which will eventually hand off to cmtk.reformatx when using CMTK.

Details

NB the sample, reference and via brains can either be templatebrain objects or a character string containing the short name of the template e.g. "IS2".

xform_brain uses the helper function shortest_bridging_seq to find the shortest path between different template brains based on the set of bridging registrations that the natverse has been informed about (see bridging_graph). You can specify a via argument to ensure that the registrations passes through one or more intermediate templates. Note that when multiple brains are passed to via they should be in order from sample to reference. If you are passing multiple templatebrain objects, they must be wrapped in a list.

When transforming image data (imagedata=TRUE), the target argument should normally be specified. This defines the absolute/voxel dimensions of the target space. This can be calculated from a templatebrain object, so by default it will be set to the value of the reference argument. Alternatively an image file on disk can be specified; this is essential if the reference argument does not specify a templatebrain object but instead just names a template space (i.e. is a string).

The significance of the imagedata and checkboth arguments is that CMTK registrations are not directly invertible although they can be numerically inverted in most cases (unless there are regions where folding occurred). For image data, numerical inversion is *much* slower.

You can control whether you want to allow inverse registrations manually by setting checkboth explicitly. Otherwise when checkboth=NULL the default is to act as if checkboth=T but issue a warning if an inversion must be used.

See Also

mirror_brain, shortest_bridging_seq bridging_graph, regtemplate, xform. xformpoints, xformimage, cmtk.reformatx (for transforming image data with CMTK).

Examples

```
## depends on nat.flybrains package and system CMTK installation
## Not run:
## reformat neurons
##
library(nat.flybrains)
# Plot Kenyon cells in their original FCWB template brain
nopen3d()
plot3d(kcs20)
plot3d(FCWB)
# Convert to JFCR2 template brain
kcs20.jfrc2=xform_brain(kcs20, sample = FCWB, reference=JFRC2)
# now plot in the new JFRC2 space
nopen3d()
plot3d(kcs20.jfrc2)
plot3d(JFRC2)
# compare with the untransformed neurons
plot3d(kcs20)
# plot with neuropil sub regions for the left mushroom body
clear3d()
plot3d(kcs20.jfrc2)
# nb "MB.*_L" is a regular expression
plot3d(JFRC2NP.surf, "MB.*_L", alpha=0.3)
# compare with originals - bridging registration is no perfect in peduncle
nopen3d()
plot3d(kcs20)
plot3d(FCWBNP.surf, "MB.*_L", alpha=0.3)

# insist on using a specific intermediate template brain
# this would nor be an improvement in this case
kcs20.jfrc2viais2=xform_brain(kcs20, sample = FCWB, via=IS2, reference=JFRC2)


## reformat image examples
# see ?cmtk.reformatx for details of any additional arguments
# note that for image data a target space defining the dimensions of the
# output image must be specified - this happens by default using the
# reference templatebrain object
xform_brain('in.nrrd', sample=FCWB, ref=JFRC2, output='out.nrrd')
# or you can specify an image file explicitly as target
xform_brain('in.nrrd', sample=FCWB, ref=JFRC2, output='out.nrrd',
            target='JFRC2.nrrd')

# use partial volume interpolation for label field
xform_brain('labels.nrrd', sample=FCWB, ref=JFRC2, output='out.nrrd',
            interpolation='pv')

# use binary mask to restrict (and speed up) reformatting
xform_brain('in.nrrd', sample=FCWB, ref=JFRC2, output='out.nrrd', mask='neuropil.nrrd')

## End(Not run)
```

Index