# Regtricks

***Release 0.3.6***

**Aug 03, 2023**

# Contents:

Tools for manipulating, combining and applying registrations.

CHAPTER 1

# Quickstart

## 1.1 Loading or creating transformations

### 1.1.1 Linear or affine registrations (eg FSL FLIRT)

API link: *regtricks.transforms.linear.Registration*

Registrations can be created from a *np.array*, a path to a text file that numpy can read, or by calling a wrapper for FLIRT. In all cases, the matrix should be 4x4 and the last row should be 0,0,0,1.

```
import regtricks as rt

# From an array
m = np.eye(4)
r = rt.Registration(m)

# From a file that numpy can read (NB if using a FLIRT matrix see below example)
p = '/a/path/to/file.txt'
r = rt.Registration(r)

# From a FLIRT matrix: provide the original source and reference images
src = 'the_source.nii.gz'
ref = 'the_reference.nii.gz'
p = 'the_flirt_matrix.mat'
r = rt.Registration.from_flirt(p, src=src, ref=ref)

# Alternatively, you can run FLIRT directly and return a Reigstration object
src = 'the_source.nii.gz'
ref = 'the_reference.nii.gz'
r = rt.flirt(src, ref, **kwargs)
```

### 1.1.2 Motion corrections (eg FSL MCFLIRT)

API link: *regtricks.transforms.linear.MotionCorrection*

Motion corrections are stored as a sequence of Registrations (eg, for a timeseries of 50 volumes, there will be 50 registrations). They can be created from a list of *np.array*, a path to a text file that shaped (4xN) x 4, a path to a folder containing *only* files for the individual arrays, or by calling a wrapper for MCFLIRT.

```
# From a list of arrays
m = [ np.eye(4) for _ in range(10) ]
mc = rt.MotionCorrection(m)

# From a file that numpy can read, shaped (4xN) x 4
p = '/a/path/to/file.txt'
mc = rt.Registration(p)

# From a directory containing individual files, named in order
p = 'a/path/to/dir'
mc = rt.MotionCorrection(p)

# From a MCFLIRT -mats directory: provide the original src and ref images
# Unless using MCFLIRT's -reffile option, the src and the ref are the same!
src = 'the_source.nii.gz'
p = '/path/to/mcflirt.mat'
mc = rt.Registration.from_flirt(p, src=src, ref=src)

# Run MCFLIRT directly and return a MotionCorrection object
src = 'the_source.nii.gz'
mc = rt.mcflirt(src, **kwargs)
```

### 1.1.3 Non-linear registrations (ie FSL FNIRT)

API link: *regtricks.transforms.nonlinear.NonLinearRegistration*

For the moment, the only way of loading in *NonLinearRegistrations* is via *FNIRT* output (or *epi_reg*, *topup*).

```
# From a FNIRT coefficients file, or displacement fields
p = '/a/path/to/fnirt.nii.gz'
src = 'src_image.nii.gz'
ref = 'ref_image.nii.gz'

# use intensity_correct = True if you want to use the Jacobian
nl = rt.NonLinearRegistration.from_fnirt(p, src, ref)
```

## 1.2 Combining and applying transformations

Transformations, of any type and in any number, can be combined into a single transformation using *rt.chain*. The order of application will be the order the transformations are given. For example, *rt.chain(A, B, C)* will apply A, then B, then C.

```
# Prepare some transformations
A = rt.Registration(some_matrix)
B = rt.MotionCorrection([some_matrices])
C = rt.NonLinearRegistration.from_fnirt(some_fnirt_file, src, ref)
```

(continues on next page)

(continued from previous page)

```
# Register, motion correct and warp, in that order
combined = rt.chain(A, B, C)

# Now apply to images
transformed = combined.apply_to_image(some_nifti)
```

## 1.3 Working with ImageSpaces (voxel grids)

API link: *regtricks.image_space.ImageSpace*

Many operations can be achieved by directly manipulating the voxel grid of an image. For example, cropping, extending, reorienting, or changing the voxel size can be achieved using methods on the *ImageSpace* object.

```
spc = rt.ImageSpace(some_nifti)

spc.resize # change dimensions of voxel grid
spc.create_axis_aligned # create a voxel grid
spc.resize_voxels # resize voxels of a grid
spc.make_nifti # make a NIFTI object from ImageSpace
spc.bbox_origin # corner of grid's bounding box
spc.touch # write empty NIFTI for ImageSpace at path
spc.voxel_centres # array of all voxel centre coordinates
spc.world2FSL # transformation from world to FSL coords
spc.world2vox # transformation from world to voxel coords
spc.FSL2world # transformation from FSL to world coords
spc.vox2world # transformation from voxel to world coords
spc.transform # transform NIFTI sform header directly
```

CHAPTER 2

# FSL integration

Regtricks can handle transformations expressed in world (aka world-mm) or FSL coordinate systems.

## 2.1 The FSL coordinate system

FSL uses a scaled-mm coordinate system. The origin of the system is always in a corner of the voxel grid and increments along each axis by the voxel dimensions. For example, if the voxel size is (1,2,3)mm, then voxel (2,2,2) will map to position (2,4,6) in FSL coordinates. For a more detailed overview of the system, see this link.

Internally, regtricks converts all FSL transformations (both linear and non-linear) into world-world convention for consistency. Regtricks will perform this conversion automatically on FSL transforms *if the appropriate functions are used*:

- `Registration.from_flirt()` for linear transforms (FSL FLIRT)
- `MotionCorrection.from_mcflirt()` for motion corrections (FSL MCFLIRT)
- `NonLinearRegistration.from_fnirt()` for non-linear transforms (FSL FNIRT, topup, epi_reg)

**Warning:** It is impossible to work out what convention a transformation is using just by inspecting it. For example, a 4x4 linear transformation matrix does not convey any information about world-world or FSL coordinate systems. The only solution is to know in advance how the transformation was generated (eg, via FSL FLIRT).

CHAPTER 3

# Contributing

Contributions are welcomed! In particular, assistance in integrating transformations generated from the ANTS, ITK or SPM packages would be welcomed. Please make contact via the GitHub repo.

CHAPTER 4

# regtricks

## 4.1 regtricks package

### 4.1.1 Subpackages

#### regtricks.transforms package

#### Submodules

#### regtricks.transforms.linear module

**class** regtricks.transforms.linear.**MotionCorrection**(*mats*)

Bases: *regtricks.transforms.linear.Registration*

A sequence of Registration objects, one for each volume in a timeseries.

**Parameters** **mats** – a path to a directory containing transformation matrices, in name order (all files will be loaded), or a list of individual filenames, or a list of np.arrays

**from_flirt**(**args*)

Load an affine (4x4) transformation between two images directly from FLIRT's -omat output.

**Parameters**

- **src2ref** (*Pathlike, np.ndarray*) – path to text-like file to load or np.ndarray
- **src** – the source of the transform
- **ref** – the reference (or target) of the transform

**Returns** Registration object

**classmethod from_mcflirt**(*mats*, *src*, *ref*)

Load a MotionCorrection object directly from MCFLIRT's -omat directory. Note that for within-timeseries registration, the src and ref arguments should take the same value.

**Parameters**

- **mats** – a path to a directory containing transformation matrices, in name order (all files will be loaded), or a list of individual filenames, or a list of np.arrays
- **src** – source of the transforms (ie, the image being corrected)
- **ref** – the target of the transforms (normally same as src)

**Returns** MotionCorrection

**classmethod from_registration**(*reg*, *length*)
Produce a MotionCorrection by repeating a Registration object n times (eg, 10 copies of a single transform)

**classmethod identity**(*length*)

**ref2src**
List of ref to src transformation matrices

**resolve**(*src*, *ref*, *at_idx*)
Return a coordinate array and scale factor that maps reference voxels into source voxels, including the transform. Uses cached values, if available.

**Parameters**

- **src** (ImageSpace) – in which data currently exists and interpolation will be performed
- **ref** (ImageSpace) – in which data needs to be expressed
- **at_idx** (*int*) – index number within series of transforms to apply

**Returns**

**(np.ndarray, 1) coordinates on which to interpolate and identity** scale factor

**save_fsl**(*outdir*, *src*, *ref*, *prefix='MAT_'*)
Save in FSL convention as textfiles at path

**save_txt**(*outdir*, *prefix='MAT_'*)
Save individual transformation matrices in text format in outdir. Matrices will be named prefix_001...

**Parameters**

- **outdir** – directory in which to save
- **src** – (optional) path to image, or ImageSpace, source space of transformation
- **ref** – as above, for reference space of transformation
- **convention** – "world" or "fsl", if fsl then src/ref must be given
- **prefix** – prefix for naming each matrix

**src2ref**
List of src to ref transformation matrices

**to_fsl**(*src*, *ref*)
Transformation matrices in FSL terms

**transforms**
List of Registration objects representing each volume of transform

**class** regtricks.transforms.linear.**Registration**(*src2ref*)
Bases: *regtricks.transforms.transform.Transform*

Affine (4x4) transformation between two images.

**Parameters** **src2ref** (*Pathlike, np.ndarray*) – path to text-like file to load or np.ndarray

**classmethod from_flirt**(*src2ref*, *src*, *ref*)
Load an affine (4x4) transformation between two images directly from FLIRT's -omat output.

**Parameters**

- **src2ref** (*Pathlike, np.ndarray*) – path to text-like file to load or np.ndarray
- **src** – the source of the transform
- **ref** – the reference (or target) of the transform

**Returns** Registration object

**classmethod identity**()

**inverse**()
Self inverse

**prepare_cache**(*ref*)
Cache re-useable data before interpolate_and_scale. Just the voxel index grid of the reference space is stored

**ref2src**

**resolve**(*src*, *ref*, **unused*)
Return a coordinate array and scale factor that maps reference voxels into source voxels, including the transform. Uses cached values, if available.

**Parameters**

- **src** (ImageSpace) – in which data currently exists and interpolation will be performed
- **ref** (ImageSpace) – in which data needs to be expressed

**Returns**

**(np.ndarray, 1) coordinates on which to interpolate and identity** scale factor

**save_fsl**(*path*, *src*, *ref*)
Save in FSL convention as textfile at path

**save_txt**(*path*)
Save as textfile at path

**src2ref**

**to_flirt**(*src*, *ref*)
Alias for self.to_fsl()

**to_fsl**(*src*, *ref*)
Return transformation in FSL convention, for given src and ref, as np.array. This will be 3D in the case of MotionCorrections

### regtricks.transforms.nonlinear module

**class** regtricks.transforms.nonlinear.**NonLinearMotionCorrection**(*warp*, *premat*, *postmat*, *intensity_correct=0*, *constrain_jac=False*)
Bases: *regtricks.transforms.nonlinear.NonLinearRegistration*

Only to be created by multiplication of other classes. Don't go here!

**Parameters**

- **warp** – FNIRTCoefficients object or NonLinearProduct
- **src** – src of transform
- **ref** – ref of transform
- **premat** – list of Registration objects
- **postmat** – list of Registration objects
- **intensity_correct** – int (0/1/2/3), whether to apply intensity correction, and at what stage in the case of NLPs
- **constrain_jac** (*bool/array-like*) – constrain Jacobian for intensity correction (default False). If True, limits of (0.01, 100) will be used, or explicit limits can be given as (min, max)

**resolve**(*src*, *ref*, *at_idx*)

Return a coordinate array and scale factor that maps reference voxels into source voxels, including the transform. Uses cached values, if available. A scale factor of 1 will be returned if no intensity correction was requested.

**Parameters**

- **src** (ImageSpace) – in which data currently exists and interpolation will be performed
- **ref** (ImageSpace) – in which data needs to be expressed
- **at_idx** (*int*) – index number within MC series of transforms to apply

**Returns**

**(np.ndarray, np.ndarray/int) coordinates on which to interpolate,** scaling factor to apply after interpolation

**class** regtricks.transforms.nonlinear.**NonLinearRegistration**

Bases: *regtricks.transforms.transform.Transform*

Non linear registration transformation. Currently only FSL FNIRT warps are supported. Note that the –premat and –postmat used by FSL command line tools should not be supplied here. Instead, defined them as Registration objects and use chain() to concatenate them with NLRs.

**classmethod from_fnirt**(*coefficients*, *src*, *ref*, *intensity_correct=False*, *constrain_jac=False*)

FNIRT non-linear registration from a coefficients file. If a pre-warp and post-warp transformation need to be applied, create these as separate Registration objects and combine them via chain, ie, combined = chain(pre, non-linear, post)

**Parameters**

- **coefficients** (*Pathlike*) – FNIRT coefficient field
- **src** (*Pathlike,* ImageSpace) – source image used for generating FNIRT coefficients
- **ref** (*Pathlike,* ImageSpace) – reference image used for generating FNIRT coefficients
- **intensity_correct** – intensity correct output via the Jacobian determinant of this warp (when self.apply_to*() is called)
- **constrain_jac** (*bool/array-like*) – constrain Jacobian for intensity correction (default False). If True, limits of (0.01, 100) will be used, or explicit limits can be given as (min, max)

**Returns** NonLinearRegistration object

**intensity_correct**

**inverse**()
Iverse warpfield, via FSL invwarp

**postmat_to_fsl**(*src*, *ref*)
Return list of postmats in FSL convention

**premat_to_fsl**(*src*, *ref*)
Return list of premats in FSL convention

**prepare_cache**(*ref*)
Pre-compute and store the displacement field, including any postmats. This is because premats can be applied after calculating the field, but postmats must be included as part of that calculation. Note that get_cache_value() return None, signifying that the field could not be cached (which implies a NLMC)

**Parameters** **ref** (*ImageSapce*) – the space in towards which the transform will be applied

**resolve**(*src*, *ref*, **unused*)
Return a coordinate array and scale factor that maps reference voxels into source voxels, including the transform. Uses cached values, if available. A scale factor of 1 will be returned if no intensity correction was requested.

**Parameters**

- **src** (ImageSpace) – in which data currently exists and interpolation will be performed
- **ref** (ImageSpace) – in which data needs to be expressed

**Returns**

**(np.ndarray, np.ndarray/int) coordinates on which to interpolate,** scaling factor to apply after interpolation

### regtricks.transforms.transform module

**class** regtricks.transforms.transform.**Transform**
Bases: object

Base object for all transformations. This should never actually be instantiated but is instead used to provide common functions.

**_cache**
use for storing resolved displacement fields and sharing amongst workers in multiprocessing pool

**islinear**
Registrations or MotionCorrections

**isnonlinear**
NonLinearRegistrations or NLMCs

**apply_to_array**(*data*, *src*, *ref*, *order=3*, *superfactor=True*, *mask=True*, *cval=0.0*, *cores=2*, ***kwargs*)
Applies transformation to data array. If a registration is applied to 4D data, the same transformation will be applied to all volumes in the series.

**Parameters**

- **data** (*array*) – 3D or 4D array.
- **src** (*Pathlike/NII/MGZ/FSLImage/ImageSpace*) – current space of data

- **ref** (*Pathlike/NII/MGZ/FSLImage/ImageSpace*) – target space for data
- **order** (*int*) – 0 for NN, 1 for linear, 2-5 for splines.
- **superfactor** (*bool/int/iterable*) – default True for any order > 0, (chosen automatically); intermediate super-sampling (replicates applywarp -super), enabled by default when resampling from high to low resolution. Set as False to disable, or set an int/iterable to manually specify level for each image dimension.
- **mask** (*bool*) – for order > 1, mask output to remove negligible values due to spline artefact
- **cval** (*float*) – fill value for background, used for correcting spline artefact
- **cores** (*int*) – CPU cores to use for 4D data
- ****kwargs** – passed on to scipy.ndimage.map_coordinates

**Returns** (np.array) transformed image data in ref voxel grid.

**apply_to_image**(*src*, *ref*, *order=3*, *superfactor=True*, *mask=True*, *cval=0.0*, *cores=2*, ***kwargs*)
Applies transformation to data array. If a registration is applied to 4D data, the same transformation will be applied to all volumes in the series.

**Parameters**

- **src** (*Pathlike/NII/MGZ/FSLImage*) – image to transform
- **ref** (*Pathlike/NII/MGZ/FSLImage/ImageSpace*) – target space for data
- **order** (*int*) – 0 for NN, 1 for linear, 2-5 for splines.
- **superfactor** (*bool/int/iterable*) – default True for any order > 0, (chosen automatically); intermediate super-sampling (replicates applywarp -super), enabled by default when resampling from high to low resolution. Set as False to disable, or set an int/iterable to manually specify level for each image dimension.
- **mask** (*bool*) – for order > 1, mask output to remove negligible values due to spline artefact
- **cval** (*float*) – fill value for background, used for correcting spline artefact
- **cores** (*int*) – CPU cores to use for 4D data
- ****kwargs** – passed on to scipy.ndimage.map_coordinates

**Returns** (np.array) transformed image data in ref voxel grid.

**cache**

**is_linear**

**is_nonlinear**

**reset_cache**()

**save**(*path*)
Save transformation at path in X5 format (experimental)

#### Module contents

### 4.1.2 Submodules

### 4.1.3 regtricks.application_helpers module

regtricks.application_helpers.**aff_trans**(*matrix*, *points*)
Affine transform a 3D set of points

regtricks.application_helpers.**despatch**(*data*, *transform*, *src_spc*, *ref_spc*, *cores*, ***kwargs*)
Apply a transform to an array of data, mapping from source space to reference. Essentially this is an extended wrapper for Scipy map_coordinates.

**Parameters**

- **data** (*array*) – np.array of data (3D or 4D)
- **transform** (*Transformation*) – between source and reference space
- **src_spc** (ImageSpace) – in which data currently lies
- **ref_spc** (ImageSpace) – towards which data will be transformed
- **cores** (*int*) – number of cores to use (for 4D data)
- ****kwargs** – passed onto scipy.ndimage.interpolate.map_coordinates

**Returns** (np.array) transformed data

regtricks.application_helpers.**interpolate_and_scale**(*idx*, *data*, *transform*, *src_spc*, *ref_spc*, ***kwargs*)
Used for partial function application to share interpolation jobs amongst workers of a mp.Pool(). Interpolate data onto the coordinates given in the tuple coords_scale, and multiply the output by the other value in coords_scale. Reshape the output to size out_size.

**Parameters**

- **data** (*np.ndarray*) – 3D, image data
- **coords_scale** (*np.ndarray, np.ndarray*) – (N,3) coordinates to interpolate onto (indices into data array), value by which to scale output (int or another np.ndarray for intensity correction)
- **out_size** (*np.ndarray*) – 3-vector, shape of output
- ****kwargs** – passed onto scipy map_coordinates

**Returns** (np.ndarray), sized as out_size, interpolated output

regtricks.application_helpers.**src_load_helper**(*src*)

regtricks.application_helpers.**sum_array_blocks**(*array*, *factor*)
Sum sub-arrays of a larger array, each of which is sized according to factor. The array is split into smaller subarrays of size given by factor, each of which is summed, and the results returned in a new array, shrunk accordingly.

**Parameters**

- **array** – n-dimensional array of data to sum
- **factor** – n-length tuple, size of sub-arrays to sum over

**Returns**

**array of size array.shape/factor, each element containing the sum of the** corresponding subarray in the input

### 4.1.4 regtricks.fnirt_coefficients module

**class** regtricks.fnirt_coefficients.**FNIRTCoefficients**(*coeffs*, *src*, *ref*, *constrain_jac=False*)

Bases: object

Private encapsulation of FNIRT warp field. Only to be used from within a NonLinearTransformation

**Parameters**

- **nibabel object or path to coefficients file** (*coeffs;*) –
- **src** – Pathlike or ImageSpace, path to original source for transform
- **ref** – Pathlike or ImageSpace, path to original reference for transform
- **constrain_jac** (*bool/tuple*) – constrain the Jacobian of the transform (default False). If True, default limits of (0.01, 100) are used, otherwise the limits (min,max) can be passed directly.

**get_cache_value**(*ref*, *postmat*)

Return cacheable values, if possible, else return None.

When can we cache? If there are only one midmat/postmat, or all of the midmats and postmats are actually the same (due to series expansion required to match the size of the premats), then we can compute and cache displacement field as it will be the same for all workers. If not, then we cannot cache and all workes must compute a new displacement field for each mid/post pair

**get_displacements**(*ref*, *postmat*, *at_idx=None*)

Resolve displacements of transform within reference space with postmat.

**Parameters**

- **ref** (ImageSpace) – space within which to resolve displacements
- **postmat** (*Registration/MotionCorrection*) – post-warp transform
- **at_idx** (*int*) – index number within postmat to use (for MC). Default None, which corresponds to Registration postmats (not MC).

**Returns**

**Nx3 array of absolute positions of reference voxels in the** grid of the warp's source space, in FSL coordinates

**Return type** (array)

**jmax**

**jmin**

**class** regtricks.fnirt_coefficients.**NonLinearProduct**(*first*, *first_post*, *second_pre*, *second*)

Bases: object

Lazy evaluation of the combination of two non-linear warps. The two warps are stored separately as FNIRTCoefficients objects, and combined into a single field via convertwarp when resolve() is called.

**Parameters**

- **first** (FNIRTCoefficients) – first warp

- **first_post** (*Registration/MotionCorrection*) – after first warp transform
- **second_pre** (*Registration/MotionCorrection*) – before second warp transform
- **second** (FNIRTCoefficients) – second warp

**get_cache_value**(*ref*, *postmat*)
Return cacheable values, if possible, else return None.

When can we cache? If there are only one midmat/postmat, or all of the midmats and postmats are actually the same (due to series expansion required to match the size of the premats), then we can compute and cache displacement field as it will be the same for all workers. If not, then we cannot cache and all workes must compute a new displacement field for each mid/post pair

**get_displacements**(*ref*, *postmat*, *at_idx=None*)
Resolve displacements of transform within reference space with postmat.

**Parameters**

- **ref** (ImageSpace) – space within which to resolve displacements
- **postmat** (*Registration/MotionCorrection*) – post-warp transform
- **at_idx** (*int*) – index number within mid/postmat to use (for MC). Default is None, which corresponds to Registration mid/postmats.

**Returns**

**Nx3 array of absolute positions of reference voxels in the** grid of the warp's source space, in FSL coordinates

**Return type** (array)

**jmax**

**jmin**

regtricks.fnirt_coefficients.**det_jacobian**(*vec_field*, *vox_size*)
Calculate determinant of Jacobian for vector field, with homogenous spacing along each axis. Second order central differences are used to estimate partial derivatives.

**Parameters**

- **vec_field** (*np.ndarray*) – sized XYZ3, where the last dimension corresponds to displacements along the x,y,z axis respectively
- **vox_size** (*np.ndarray*) – array sized 3, step size along each spatial axis

**Returns**

**(np.ndarray), sized XYZ, values of the determinant of the Jacobian** matrix evaluated at each point within the array

regtricks.fnirt_coefficients.**get_field**(*coeff1*, *ref*, *coeff2=None*, *mid=None*, *post=None*, *jmin=None*, *jmax=None*)
Resolve coefficients into displacement field via convertwarp.

**Parameters**

- **coeff1** (FNIRTCoefficients) – first warp
- **ref** (ImageSpace) – reference grid for output
- **coeff2** (FNIRTCoefficients) – optional, second warp
- **mid** (*np.ndarray*) – optional, between-warp affine matrix

- **post** (*np.ndarray*) – optional, after-warp affine matrix

**Returns**

**np.ndarray, shape Nx3, arranged by voxel index down the rows and** XYZ in columns. Row M in the array gives the position of the reference voxel with linear index M in the *source* voxel grid of warp1, *in FSL coordinates*.

### 4.1.5 regtricks.image_space module

ImageSpace: image matrix, inc dimensions, voxel size, vox2world matrix and inverse, of an image. Used for resampling operations between different spaces and also for saving images into said space (eg, save PV estimates into the space of an image)

**class** regtricks.image_space.**ImageSpace**(*img*)

Bases: object

Voxel grid of an image, ignoring actual image data.

**Parameters** **img** – Pathlike to image, nibabel Nifti/MGH or FSL Image object

**size**
array of voxel counts in each dimension

**vox_size**
array of voxel size in each dimension

**vox2world**
4x4 affine to transform voxel coords -> world

**world2vox**
inverse of above

**FSL2vox**
Transformation from FSL scaled coordinates to voxels

**FSL2world**
Transformation from FSL scaled coordinates to world

**bbox_origin**
Origin of the image's bounding box, referenced to first voxel's corner, not center (ie, -0.5, -0.5, -0.5)

**classmethod create_axis_aligned**(*bbox_corner*, *size*, *vox_size*)
Create an ImageSpace from bounding box location and voxel size. Note that the voxels will be axis-aligned (no rotation).

**Parameters**

- **bbox_corner** – 3-vector, location of the furthest corner of the bounding box, at which the corner of voxel 0 0 0 will lie.
- **size** – 3-vector, number of voxels in each spatial dimension
- **vox_size** – 3-vector, size of voxel in each dimension

**Returns** ImageSpace object

**file_name**

**fov_size**
FoV associated with image, in mm

**ijk_grid**(*indexing='ij'*)
Return a 4D matrix of voxel indices for this space. Default indexing is 'ij' (matrix convention), 'xy' can also be used - see np.meshgrid for more info.

**Returns**

**4D array, size of this space in the first three dimensions, and** stacked I,J,K in the fourth dimension

**make_nifti**(*data*)
Construct nibabel Nifti for this voxel grid with data

**classmethod manual**(*vox2world*, *size*)
Manual constructor

**n_vox**
Number of voxels in grid

**resize**(*start*, *new_size*)
Resize the FoV of this space, maintaining axis alignment and voxel size. Can be used to both crop and expand the grid. For example, to expand the grid sized X,Y,Z by 10 voxels split equally both before and after each dimension, use (-5,5,5) and (X+5, Y+5, Z+5)

**Parameters**

- **start** – sequence of 3 ints, voxel indices by which to shift first voxel (0,0,0 is origin, negative values can be used to expand and positive values to crop)
- **new_size** – sequence of 3 ints, length in voxels for each dimension, starting from the new origin

**Returns** new ImageSpace object

**resize_voxels**(*factor*, *mode='floor'*)
Resize voxels of this grid.

**Parameters**

- **factor** – either a single value, or 3 values in array-like form, by which to multiply voxel size in each dimension
- **mode** – "floor" or "ceil", whether to round the grid size up or down if factor does not divide perfectly into the current size

**Returns** new ImageSpace object

**save_image**(*data*, *path*)
Save 3D or 4D data array at path using this image's voxel grid

**classmethod save_like**(*ref*, *data*, *path*)
Save data into the space of an existing image

**Parameters**

- **ref** – path to image defining space to use
- **data** – ndarray (of appropriate dimensions)
- **path** – path to write to

**touch**(*path*, *dtype=<class 'float'>*)
Save empty volume at path

**transform**(*reg*)
Apply affine transformation to voxel grid of this space. If the reg is a np.array, it must be in world-world terms, and if it is a Registration object, the world-world transform will be used automatically.

**Parameters** **reg** – either a 4x4 np.array (in world-world terms) or Registration

**Returns** a transformed copy of this image space

**vox2FSL**
Transformation between voxels and FSL coordinates (scaled mm). FLIRT matrices are given in (src FSL) -> (ref FSL) terms. See: https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FLIRT/FAQ

**vox_size**
Voxel size of image

**voxel_centres**(*indexing='ij'*)
Return a 4D matrix of voxel centre coordinates for this space. Default indexing is as for ImageSpace.ijk_grid(), which is 'ij' matrix convention. See np.meshgrid for more info.

**Returns**

**4D array, size of this space in the first three dimensions, and** stacked I,J,K in the fourth dimension.

**world2FSL**
Transformation from world coordinates to FSL scaled

**world2vox**
World coordinates to voxels

### 4.1.6 regtricks.multiplication module

Functions for combining Transformations

regtricks.multiplication.**cast_potential_array**(*arr*)
Helper to convert 4x4 arrays to Registrations if not already

regtricks.multiplication.**chain**(**args*)
Concatenate a series of transformations (Registration, MotionCorrection, NonLinearRegistration). Note that intensity correction should be enabled when creating a NonLinearRegistration object using intensity_correct=True in the constructor prior to using chain().

**Parameters** ***args** – Transform objects, given in the order that they need to be applied (eg, for A -> B -> C, give them in that order and they will be multiplied as C @ B @ A)

**Returns** Transform object representing the complete transformation

regtricks.multiplication.**get_highest_type**(*first*, *second*)
When combining two arbitrary transforms, the output will be of the same type as the "highest" of the two arguments (this is the "type promotion"). This function returns the highest type of two input objects, according to:

Registration LOWEST MotionCorrection NonLinearReigstration NonLinearMotionCorrection HIGHEST

Once the higest type is known, the actual multiplication is handled by that class' invididual method, as below in this submodule

**Parameters**

- **first** (*transformation*) – order doesn't matter
- **second** (*transformation*) – order doesn't matter

**Returns** type object of the highest class

regtricks.multiplication.**moco**(*lhs*, *rhs*)
Combine either a Registration and MoCo, or two MoCos. Return a MotionCorrection.

regtricks.multiplication.**nonlinearmoco**(*lhs*, *rhs*)
Combine either a Registration and NLMC, a MoCo and NLMC, a NLR and NLMC, or two NLMCs. Note at most 2 non-linear transforms can be combined. Return a NonLinearMotionCorrection.

regtricks.multiplication.**nonlinearreg**(*lhs*, *rhs*)
Combine either a Registration and NLR, a MoCo and NLR, or two NLRs. Note at most 2 non-linear transforms can be combined. Return a NonLinearRegistration.

regtricks.multiplication.**registration**(*lhs*, *rhs*)
Combine two Registrations, return a Registration.

### 4.1.7 regtricks.wrappers module

regtricks.wrappers.**flirt**(*src*, *ref*, ***kwargs*)
FLIRT registration wrapper. If any of the output arguments are given (out/o and omat), FLIRT will run in command line mode and save the outputs at those paths, and nothing will be returned. If none of the outputs are given, no outputs will be saved and a Registration will be returned. See fsl.wrappers.flirt.flirt for full docs.

**Parameters**

- **src** – image to register
- **ref** – target to register on to
- **out/o** – where to save output
- **omat** – save registration matrix
- ****kwargs** – as for FLIRT

**Returns** Registration object

regtricks.wrappers.**fnirt**(*src*, *ref*, ***kwargs*)

regtricks.wrappers.**mcflirt**(*src*, *refvol=-1*, ***kwargs*)
MCFLIRT motion correction wrapper. If an output path is given, MCFLIRT will run in command line mode, save the output and return None. If no output path is given, a MotionCorrection and Nibabel image for the frame used as the reference will be returned. See fsl.wrappers.flirt.mcflirt for full docs.

**Parameters**

- **src** – (Pathlike) image to register
- **refvol** – target frame to register on to, default is N/2
- **out** – where to save output
- ****kwargs** – as for MCFLIRT

**Returns** MotionCorrection object

### 4.1.8 regtricks.x5_interface module

X5 interface for regtricks. With thanks to Paul McCarthy; this is almost a direct copy of his fslpy.transform.x5 module

**exception** regtricks.x5_interface.**X5Error**
Bases: Exception

regtricks.x5_interface.**check_is_x5**(*path*)

regtricks.x5_interface.**load_manager**(*path*)
Load transformation objects from X5 format

regtricks.x5_interface.**read_affine**(*group*)
Load a single or stack of (4,4) arrays to X5 group

regtricks.x5_interface.**read_imagespace**(*group*)
Read ImageSpace properties (size, voxel size, vox2world) into X5 format, and return ImageSpace object

regtricks.x5_interface.**read_metadata**(*group*)
Read X5 format metadata

regtricks.x5_interface.**save_manager**(*reg*, *path*)
Save Registration or MotionCorrection objects in X5 format

regtricks.x5_interface.**write_affine**(*group*, *matrix*, *inverse*)
Write a single or stack of (4,4) arrays to X5 group

regtricks.x5_interface.**write_imagespace**(*group*, *spc*)
Write ImageSpace properties (size, voxel size, vox2world) into X5 format

regtricks.x5_interface.**write_metadata**(*group*)
Write X5 format metadata

### 4.1.9 Module contents

CHAPTER 5

# Index

# CHAPTER 6

## What is it?

Regtricks is a python library that simplifies the process of working with and applying image registrations. **It is not** a replacement for image registration tools (eg FSL FLIRT), but it does make working with the output of these tools eaiser. It offers:

- an **object oriented** interface: *Registration*, *MotionCorrection*, *NonLinearRegistration*
- easy **combining of multiple transforms** via *chain()*
- **one interpolation step** for image data, regardless of the number of transforms that need to be applied
- **separation between transformation and interpolation**: generate a registration using one pair of images, then apply it to another, even if the voxel grids don't match
- **full support** for FSL tools `FLIRT`, `MCFLIRT`, `FNIRT`
- **intensity correction** via the Jacobian determinant for *NonLinearRegistration*
- **multi-core support** for 4D images
- **lazy evaluation** avoids locking up your process until its time to actually transform an image
- full **control over interpolation**, including order of spline interpolant and pre-filtering
- an *ImageSpace* **class for manipulating voxel grids** directly (eg cropping and supersampling)

CHAPTER 7

# Why?

## 7.1 Transformation, not interpolation

Many registration tools (eg FSL) combine image *transformation* and *interpolation* into a single step, although they are **not** the same thing.

A transformation moves an image so that it is aligned with some other image (eg, functional with structural). An interpolation redraws an image from one voxel grid onto another (eg, functional to structural resolution).

What if you want to do one, but not the other? For example, transform a functional image into alignment with a structural, but leave the result in functional resolution? Regtricks separates the two operations from each other, allowing you to apply whatever transformations you want, and then choose which voxel grid in which to place the results.

## 7.2 Minimise interpolation

Regtricks allows multiple transformations to be combined with a **single** interpolation step. This preserves image quality by minimising interpolation-induced blurring.

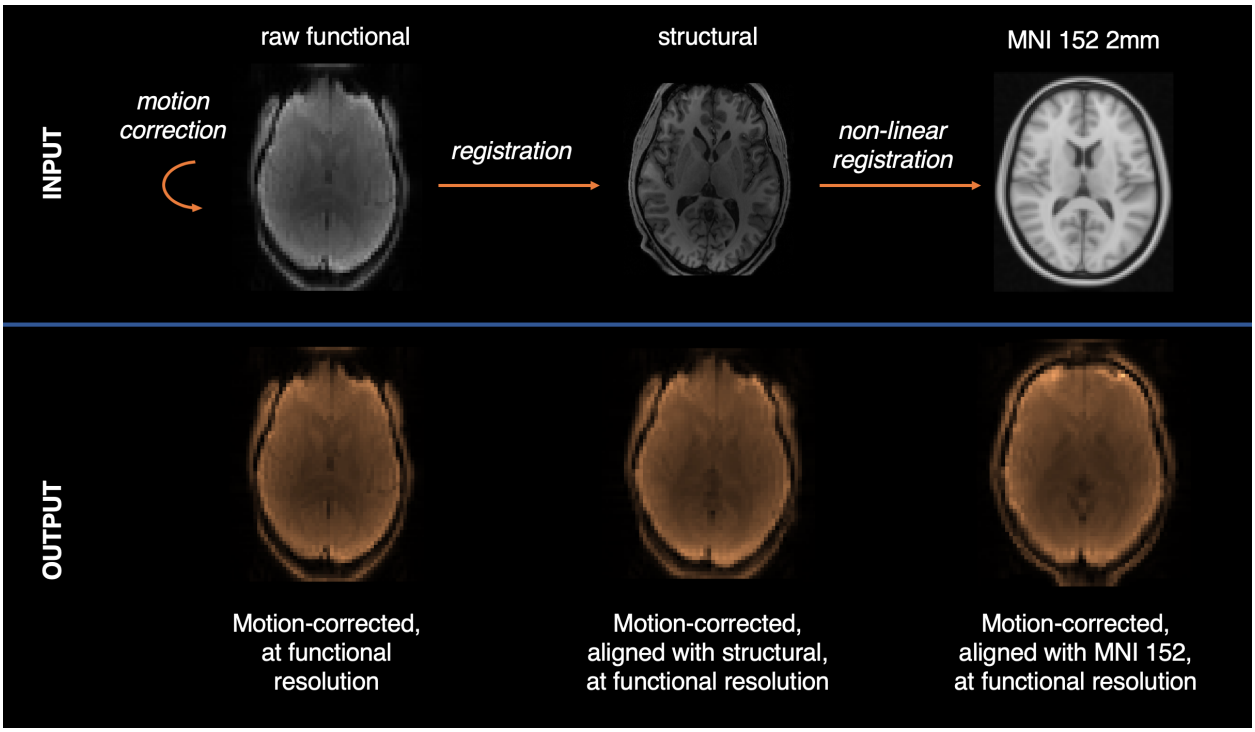


Fig. 1: Example usage of regtricks, showing the difference between transformation and interpolation. From the images in the top row, three *transformations* are generated: a motion correction, a registration, and a non-linear registration. On the bottom row, the transformations are applied, but the result is left *in the space of the original input* (functional resolution).

# CHAPTER 8

# How does it work?

## 8.1 Transformations

The subclasses of `Transformation` represent all the different types of registration:

- *`Registration`*: a linear affine registration (4x4 matrix)
- *`MotionCorrection`*: a linear motion correction (series of 4x4 matrices)
- *`NonLinearRegistration`*: a non-linear registration (aka warp, only FSL FNIRT currently supported)

All of these objects can be combined together, either via @ multiplication (NB reverse order, eg BC @ AB = AC), or *much* simpler: the *`chain()`* method! For example, if you want to motion correct a functional image and register it onto a standard space in a single step:

```
import regtricks as rt

# Load MCFLIRT, FLIRT and FNIRT transformations for each step
func_mc = rt.MotionCorrection.from_mcflirt('func_mcf.mat', src='func.nii.gz', ref=
↪'func.nii.gz')
func2struct = rt.Registration.from_flirt('func2struct.mat', src='func.nii.gz', ref=
↪'struct.nii.gz')
struct2mni = rt.NonLinearRegistration.from_fnirt('struct2mni_coeff.nii.gz', src=
↪'struct.nii.gz', ref='mni.nii.gz')

# Combine them into a single transformation that maps functional to MNI
func2mni_mc = rt.chain(func_mc, func2struct, struct2mni)

# Apply them to get a nibabel NIFTI object back:
func_mni = func2mni_mc.apply_to_image('func.nii.gz', ref='mni.nii.gz')
nibabel.save(func_mni, 'func_mni.nii.gz')
```

Regtricks features *type promotion*. For example, if you chain a *`Registration`* and a *`MotionCorrection`* together, the result is a new *`MotionCorrection`*. This applies for all transform classes and requires no user action. All transform classes have an `inverse()` method that returns the self-inverse as a new object.

## 8.2 Image spaces

All registration operations in regtricks are applied in two stages:

1. apply a transformation to move the input image
2. write out the result on some voxel grid

Although there is only a single way of doing step (1), there are many ways of doing step (2): do you want the result in the space of the input image, in the space of the target image, or in some other space entirely (eg MNI)? This is where the *ImageSpace* class comes in.

The *ImageSpace* class is used to represent the voxel grid of an image (ie, field of view, voxel size, position in world space). Although you probably won't need to interact with it directly, its handy to know why it exists. *ImageSpace* are used to denote where image data has come *from* and where it is going *to*. Almost all regtricks functions or classes accept a `src` and `ref` argument which represent the *from* and *to* respectively.

## 8.3 Applying

Transformations are applied with the `apply_to_image(src, ref)` method, where `src` is the input image and `ref` is the space in which to place the output (which could be the same as `src`). This function also accepts numerous extra arguments, for example:

- `superfactor`: intermediate super-sampling factor (similar to FSL applywarp)
- `order`: order of spline interpolant, 1-5, default 3
- `cores`: multi-core processing to speed up 4D images
- `**kwargs`: any args accepted by `map_coordinates()`

Note that intensity correction via Jacobian determinants for non-linear transforms can be applied, but it must be set when creating a non-linear transform object, not when calling `apply_to_image()`. For example:

```
epi_distcorr = NonLinearRegistration(*, intensity_correct=True).
```

Scipy's `map_coordinates()` is used to perform interpolation; this is a powerful tool that accepts many extra arguments (passed via `**kwargs`).

## 8.4 FSL Wrappers

Wrappers for the standard FSL registration functions are available in `regtricks.wrappers`. These behave slightly differently to normal commandline tools in that they return transformation objects. For example: `a_flirt = rt.wrappers.flirt(src, ref)` will run FLIRT and output a *Registration* object directly.

# Python Module Index

## r

# Index

## Symbols



## A



## B



## C



## D



## F



## G



## I

## J



## L



## M



## N



## P



## R

## S



## T



## V



## W



## X