Regtricks

Release 0.3.3.post5

Mar 17, 2021

Contents:

Tools for manipulating, combining and applying registrations.

CHAPTER 1

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, and apply them to another, even if they are in different spaces/resolutions
- **full support** for `FSL FLIRT`, `MCFLIRT`, `FNIRT`
- **intensity correction** via the Jacobian determinant for `NonLinearRegistration`
- **multi-core support** for 4D images
- 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 2

Why?

2.1 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.

2.2 Minimise interpolation

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

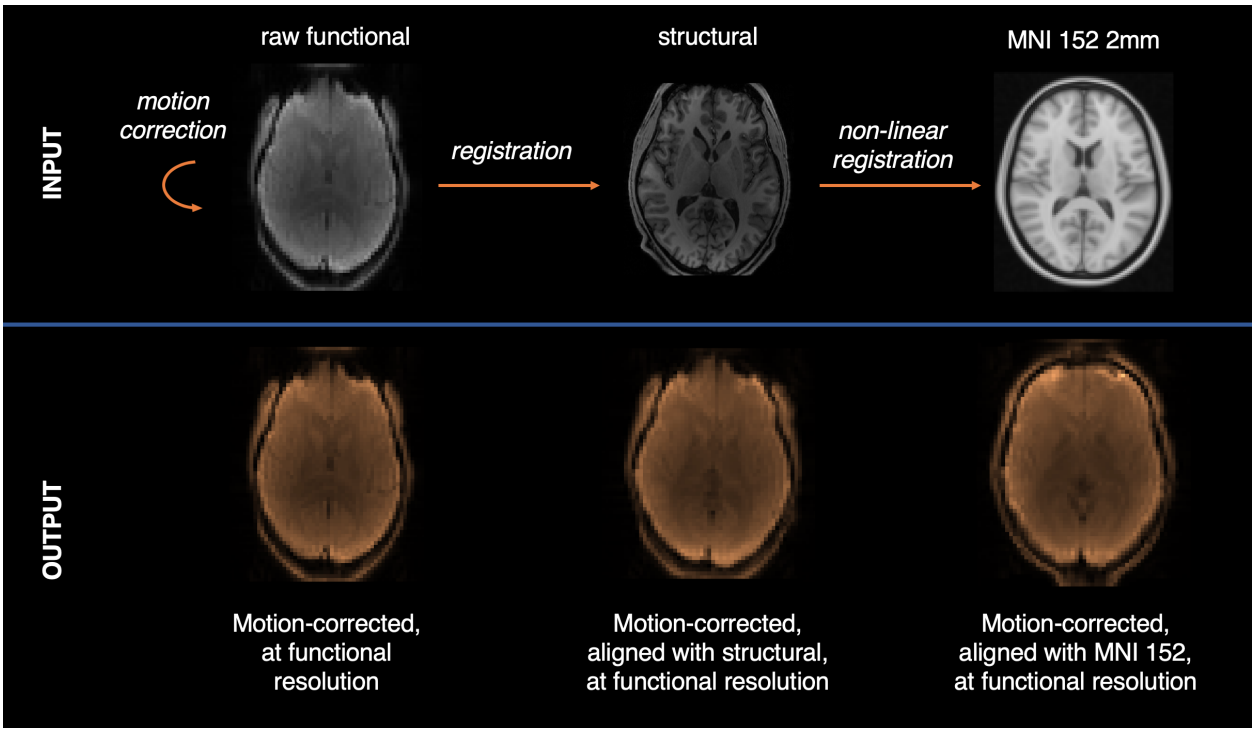

Fig. 1: Example usage of regtricks, showing the difference between transformation and interpolation. From the images in the top row, three *transformations* are generated: a motion correction, a registration, and a non-linear registration. On the bottom row, the transformations are applied, but the result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

CHAPTER 3

How does it work?

3.1 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.

3.2 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. `ImageSpaces` 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.

3.3 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`).

3.4 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.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search