SOFTWARE TOOL TO READ, REPRESENT, MANIPULATE, AND APPLY N-DIMENSIONAL SPATIAL TRANSFORMS

O. Esteban[1], M. Goncalves[1], C. J. Markiewicz[1], S. S. Ghosh[2], and R. A. Poldrack[1]

[1]Dept. of Psychology, Stanford University
[2]McGovern Institute for Brain Research, MIT and Dept. of Otolaryngology, Harvard Medical School

ABSTRACT

Spatial transforms formalize mappings between coordinates of objects in biomedical images. Transforms typically are the outcome of image registration methodologies, which estimate the alignment between two images. Image registration is a prominent task present in nearly all standard image processing and analysis pipelines. The proliferation of software implementations of image registration methodologies has resulted in a spread of data structures and file formats used to preserve and communicate transforms. This segregation of formats hinders the compatibility between tools and endangers the reproducibility of results. We propose a software tool capable of converting between formats and resampling images to apply transforms generated by the most popular neuroimaging packages and libraries (AFNI, FSL, FreeSurfer, ITK, and SPM). The proposed software is subject to continuous integration tests to check the compatibility with each supported tool after every change to the code base (https://github.com/poldracklab/nitransforms). Compatibility between software tools and imaging formats is a necessary bridge to ensure the reproducibility of results and enable the optimization and evaluation of current image processing and analysis workflows.

Index Terms— image registration, spatial transforms, software infrastructure, BIDS

1. INTRODUCTION

During the development of *fMRIPrep* [1], we found the most challenging issues to arise from the large proliferation of software tools implementing image processing and analysis methods. *fMRIPrep* is a robust pre-processing pipeline for functional magnetic resonance imaging (fMRI) that executes several image registration tools from neuroimaging packages such as AFNI [2], FSL [3], FreeSurfer [4], and ANTs [5]. While all these packages meet some minimum coverage in the implementation of the NIfTI format that ensures certain compatibility for imaging data between them, they all generate incompatible objects and file formats to describe spatial transforms. For example, the very popular ANTs registration tools are based on the Insight Toolkit (ITK), and thus express all transforms in a right-handed, continuous coordinate system with physical units (e.g., mm). FSL, on the other hand, utilizes an intermediate, left-handed coordinate system, whose interpretation is influenced by the origin and destination coordinate systems. In other words, to correctly interpret a linear transform estimated with FSL tools, it is necessary to read the orientation affine matrices of *reference* and *moving* images. A large development effort in *fMRIPrep* was directed to overcome compatibility issues, and particularly those generated by these differences in the implementation of spatial transforms.

Compatibility and exchangeability issues derived from neuroimaging data formats and structures have been previously addressed. NiBabel [6] provides a Python implementation of tools to read, write and handle data in a range of 3D image formats, as well as surface formats and mixed volume and irregular points formats. More recently, the Brain Imaging Data Structure [7, BIDS] prescribes how neuroimaging data and metadata should be stored and preserved at the study level. BIDS specifications are currently being extended along several lines. In particular, the BIDS-Derivatives extension deals with the organization and naming conventions of the final product of processing and analysis pipelines to ensure their shareability. The proposed software tool will be distributed as a new module within the NiBabel package, although it was originally conceived and designed as an offspring of the BIDS-Derivatives specifications for spatial transforms.

2. MATERIALS AND METHODS

2.1. Spatial transforms and image alignment

Let $\mathbf{x}$ represent the coordinates of a point in the reference coordinate system $\mathcal{F}$, and $\mathbf{x}'$ its projection on to another coordinate system $\mathcal{M}$:

$$\begin{aligned} \mathcal{F} \subset \mathbb{R}^n \rightarrow \mathcal{M} \subset \mathbb{R}^n \\ \mathbf{x} \mapsto \mathbf{x}' = f(\mathbf{x}). \end{aligned} \tag{1}$$

This work was supported by the Laura and John Arnold Foundation, the NIH (NBIB R01EB020740), NIMH (R24MH114705 and R24MH117179).

In an image registration problem, $\mathcal{M}$ is a moving image from which we want to sample data in order to bring the image into spatial alignment with the reference image $\mathcal{F}$ (or "fixed" image). Hence, f here is the spatial transformation function that maps from coordinates in $\mathcal{F}$ to coordinates in $\mathcal{M}$. There are a multiplicity of image registration algorithms and corresponding image transformation models to estimate linear and nonlinear transforms.

The problem has been traditionally confused by the need of *transforming* or mapping one image (generally referred to as *moving*) into another that serves as reference, with the goal of *fusing* the information from both. An example of image fusion application would be the alignment of functional data from one individual's brain to the same individual's, corresponding anatomical MRI scan for visualization. Therefore, "applying a transform" entails two operations: first, transforming the coordinates of the samples in the fixed image $\mathcal{F}$ to find their mapping $\mathbf{x}'$ on $\mathcal{M}$ via f, and second, an interpolation step as $\mathbf{x}'$ will likely fall off-the-grid of the moving image $\mathcal{M}$. Counter-intuitively, while the spatial transformation projects from $\mathcal{F}$ to $\mathcal{M}$, the voxel data flows in the opposite direction after the interpolation of values of $\mathcal{M}$ at the mapped coordinates $\mathbf{x}'$ (Figure 1).

2.2. Linear and nonlinear transforms

For the case of linear transforms, the mathematical operation f can be described as the dot product of an affine matrix A and the input vector $\mathbf{x}$ given in homogeneous coordinates:

$$\begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \\ 1 \end{bmatrix} = A \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix},$$

where A is an $(n+1, n+1)$ matrix. When the transform is nonlinear, f in (1) can be rewritten as: $f(\mathbf{x}) = \mathbf{x} + \mathbf{u}(\mathbf{x})$, where $\mathbf{u}(\mathbf{x})$ can be a nonparametric map of displacements (or "*displacements field*") or a parametric function. A widely used parametric basis to support nonlinear transforms are tensor product B-Splines. In the following example, tensor product B-Spline bases of degree three β_3 are applied: $f(\mathbf{x}) = \mathbf{x} + \beta_3(\mathbf{x})$. For numerical stability reasons, one can be interested in including the original position $\mathbf{x}$ within the mapping function. Some authors have referred to this implementation decision as "*deformation*" fields (nonparametric case) or models (parametric case): $f(\mathbf{x}) = U(\mathbf{x})$.

2.3. Composition and inversion of transforms

Transforms are generally composed sequentially to achieve the desired transformation. A typical example is first approximating images via a linear registration step followed by a nonlinear step. In such a case, (1) can be rewritten for a composite transform C as: $C(\mathbf{x}) = \mathbf{x}' = f_{\text{linear}}(f_{\text{nonlinear}}(\mathbf{x}))$.

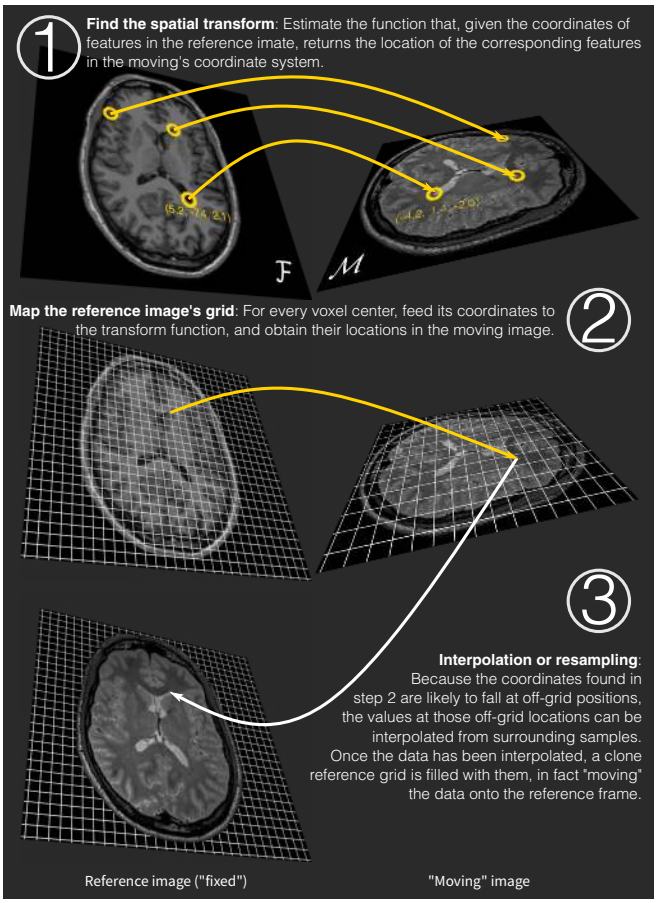

Fig. 1. Resampling the moving image onto the fixed image via a spatial transform.

Finally, a useful operation to handle transforms is inversion:

$$\begin{aligned} \mathcal{M} \subset \mathbb{R}^n \rightarrow \mathcal{F} \subset \mathbb{R}^n \\ \mathbf{x}' \mapsto \mathbf{x} = f^{-1}(\mathbf{x}'). \end{aligned} \tag{2}$$

For linear transforms, there is an analytical inversion of f. However, for nonlinear transforms, only a numerical approximation might be possible.

2.4. Software architecture

The proposed tool is based on NiBabel, and complements it for the spatial transforms. Initially, it is being developed open-source (`https://github.com/poldracklab/nitransforms`), as an isolated module, which will be integrated and distributed with NiBabel. The module is designed to require minimal dependencies beyond those already present in NiBabel. Three main components sustain the division of functional requirements: a `base` submodule where most abstract classes are defined, a `linear` submodule implementing n-dimensional linear transforms, and a `nonlinear` submodule for both parametric and non-

parametric nonlinear transforms. To ensure the consistency and uniformity of internal operations, all transforms are defined using a left-handed coordinate system of physical coordinates. In words from the neuroimaging domain, the coordinate system of transforms is *RAS+* (or positive directions point to the Right-hand for the first axis, Anterior for the second, and Superior for the third axis). The internal representation of transform coordinates is the most relevant design decision, and implies that a conversion of coordinate system is necessary to correctly interpret transforms generated by other software (e.g., ITK, where the internal coordinate system is *LPS+*, or positive on Left, Posterior, and Superior).

An additional requirement is to provide an easy interface for researchers to 1) map point sets via transforms; and 2) apply transforms (i.e., mapping the coordinates and interpolating the data) to data structures seamlessly (Figure 1). In addition to the generally easy toolset available to apply transforms onto NIfTI images, the tool also provides convenient methods to apply transforms to point sets or surfaces or mixed types combining point sets/surfaces/regularly gridded points (supporting the GIFTI format). An example of particular interest of the latter category (combining surface and volume sampling) are "grayordinates" files (see the CIFTI format).

2.5. Software tests and validation

The proliferation of existing tools and their internal implementations makes software testing and validation the most important element of this study. Using a test-driven development paradigm, we first wrote the battery of tests the software will need to pass. The tests are included in a continuous integration framework which assesses the continuity of the implementation along the development life, ensuring that new features and code changes do not break existing functionalities. We implemented two categories of tests: unit tests and cross-tool comparison tests. Unit tests evaluate the formal correctness of the implementation, to check the robustness of the software across data inputs and the coverage of all the functional requirements with an easy API (application programmer interface). Cross-tool comparison tests assess the correct implementation of third-party software, and are based on two test oracles:

- Loading transforms of software S: applying the same transform file, given in the format of one particular neuroimaging tool S, to a pair of images should arrive to the same resulting image if nearest-neighbor interpolation is used.

- Writing transforms for software S: consistently, generating some transform and writing it to a particular software S format should generate the same interpolated images if nearest-neighbor interpolation is used.

3. RESULTS

3.1. Software implementation

As a result of the design proposed in subsection 2.4, software contributions are released including the following salient features:

An easy-to-use, comprehensive interface. The software follows NiBabel's standards for the input/output interface, providing a global `load()` function to read in transform files written by any of the supported software tools, and a `<transform_object>.to_filename()` function that writes the transform object out to hard disk. Transform objects have a function call implementing f in (1).

```
import nitransforms as nt
f = nt.io.load("from-FSL.txt")
x_prime = f([10.0, -5.0, 2.0])
```

The transform object also provides an `apply()` method to resample any given image, surface/point-set or hybrid (volume and surface) object via the transform.

```
f = nt.io.load("from-FSL.txt",
               reference="ref.nii")
f.apply("moving.nii").to_filename(
    "moved-to-ref.nii.gz")
f.apply("moving.nii",
        reference="surface.gii"
).to_filename("moved-to-surf.gii")
```

Finally, the composition and inversion operations are also available. For the case of composition, let `f1` be a linear transform and `f2` a nonlinear refinement that should be concatenated (please note that the concatenation operation `+` requires operators to be ordered; i.e., it's non-commutative):

```
(f1 + f2).apply("moving.nii",
        reference="surface.gii"
).to_filename("moved-to-surf.gii")
```

Since transforms are converted into the internal representation at loading time, composition allows the combination of transforms from multiple software packages. Inversion (2) is implemented following an API similar to that of the NumPy package:

```
f_inv = nt.inv(f, reference="moving.nii",
               approx=True)
f_inv.apply("reference.nii").to_filename(
    "apply-inverted.nii")
```

Tests coverage. Tests coverage is broadly defined as the percentage of lines of code that have been exercised (executed) at least once during some of the tests. Although this metric does not guarantee any theoretical correctness of the

implementation, it is a good indicator of sections of the implementation that should be more thoroughly tested. The project has a 98% coverage.

Linting and other forms of validation. The software code adheres to the current standards in code style recommended by Python, and code is linted (i.e., formally validated) with every change to the code base. Linting checks the syntax of code, and applies a series of rules and heuristics to highlight potential points of fault. For instance, defining a variable that is seemingly never used.

3.2. Compatibility with existing software instruments

A test set checks that transforms written by any of the neuroimaging packages supported (AFNI, FSL, FreeSurfer, and ITK) are correctly loaded and interpreted, ensuring that the result of applying such a transform is the same as, within an acceptable margin of error, that obtained using the original tool. Similarly, we also test that transforms generated internally by the test battery are correctly exported to the formats of the above mentioned tools, and that the results of transforming an image with the proposed software and the supported tools are equivalent within the margin of error. These tests are run with every code change to ensure that functionality is preserved throughout maintenance and development cycles.

4. DISCUSSION

We present a piece of software that will be fundamental to those trying to create reproducible neuroimaging pipelines that make use of several available toolboxes. Such an endeavour was nearly impossible before the NIfTI standard existed, but spatial transforms have remained uncovered by any common solution across packages. Although we do not propose any purely theoretical or methodological contribution, the software instruments proposed here are currently lacking and crucial to enable single-step, uncomplicated resampling of images and surfaces in pipelines like *fMRIPrep*. On the methodological aspect, we propose that infrastructural contributions as ours must be supported by thorough, continuous testing, stringent formal checks (e.g., language linting), and extensive documentation. The project has been driven by these software engineering "good practices" from the start, to ensure the highest standards for the implementation. As if it were a preregistration of an experiment, we first wrote the tests that assess whether all functional requirements have been met, and then initiated the implementation (test-driven development).

The software is built as a base for the larger effort in standardization of the BIDS-Derivatives, an extension of BIDS to describe the outcomes of processing and analysis pipelines. In particular, this software instrumentation is conceived as part of the BIDS Extension Proposal 14 for linear and non-linear spatial mappings.

A limitation of the software is its potential confinement to neuroimaging tools and formats. However, generalization to other domains is straightforward considering that the theoretical foundations are shared. Implementation-wise, the software finds a limitation when applying surface-to-surface transforms on the surface's manifold (typically conducted on the sphere). Future versions of the tool will implement this feature.

The field of neuroimaging is experiencing a paradigm shift, where the knowledge of computational methods required of scientists is approaching that of the required neuroscience domain knowledge. Therefore, it is essential to engineer new software instruments that help unload the methodological details of software implementation without creating black-boxes that scientists operate blindly. By making it conceptually clearer, with an easy-to-use interface, the proposed software will increase the reliability of scientific findings, enable a more thorough assessment and comparison across tools, and maximize the shareability and reproducibility of the results.

5. REFERENCES

[1] O. Esteban et al., "fMRIPrep: a robust preprocessing pipeline for functional MRI," *Nat Meth*, vol. 16, no. 1, pp. 111–116, 2019.

[2] R. W. Cox and J. S. Hyde, "Software tools for analysis and visualization of fmri data," *NMR Biomed*, vol. 10, no. 4-5, pp. 171–178, 1997.

[3] M. Jenkinson et al., "FSL," *NeuroImage*, vol. 62, no. 2, pp. 782–790, 2012.

[4] B. Fischl, "FreeSurfer," *NeuroImage*, vol. 62, no. 2, pp. 774–781, 2012.

[5] B. Avants et al., "Symmetric diffeomorphic image registration with cross-correlation: Evaluating automated labeling of elderly and neurodegenerative brain," *Med Image Anal*, vol. 12, no. 1, pp. 26–41, 2008.

[6] M. Brett et al., "nipy/nibabel: 3.0.0," *Zenodo*, Dec. 2019.

[7] K. J. Gorgolewski et al., "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments," *Sci Dat*, vol. 3, pp. 160044, 2016.