Converting ANTs affine matrix to a 4×4 homogeneous matrix

Kaibo Tang

February 10, 2024

1 Background

Affine registration between volumes is a common procedure in a medical image pre-processing pipeline. **ANTs**[1] is a popular and clinically validated toolbox that we use frequently. Performing affine registration using ANTs is fairly straight forward. We assume readers of this note have background knowledge of ANTs.

For the rest of this note, suppose we are using ANTsPy[2] and we use the following function to perform affine registration on the moving and fixed images.

```python
import ants
import shutil

def run_ants_affine(moving_fn: str, fixed_fn: str) -> None:
    moving = ants.image_read(moving_fn)
    fixed = ants.image_read(fixed_fn)
    output = ants.registration(fixed=fixed, moving=moving,
     ↪   type_of_transform='Affine')
    # save warped image
    ants.image_write(output['warpedmovout'], 'affine.nii.gz')
    # save forward transformation
    shutil.copyfile(output['fwdtransforms'][0], 'affine.mat')
```

[1]https://github.com/ANTsX/ANTs
[2]https://github.com/ANTsX/ANTsPy

The output file `affine.nii.gz` is the transformed (or warped) image while `affine.mat` is the associated "affine matrix", which defines how the transformation is performed under **world coordinates**.

2 Problem

When we load the `.mat` file using YFP (your favorite package), we see that the content of the file actually looks something like this.

```
{'AffineTransform_float_3_3': array([[ 9.67558980e-01],
       [ 3.71180326e-02],
       [ 1.23559460e-02],
       [-4.93900105e-02],
       [ 9.07873511e-01],
       [-2.29182512e-01],
       [-3.32694985e-02],
       [ 2.48323038e-01],
       [ 8.58248472e-01],
       [ 4.41851616e-02],
       [ 5.25157928e-01],
       [-1.46762085e+01]], dtype=float32),
       'fixed': array([[126.26413],
       [127.34694],
       [134.72673]], dtype=float32)}
```

It's a dictionary containing an `ndarray` of shape `(12, 1)` whose key is `'AffineTransform_float_3_3'` and an `ndarray` of shape `(3, 1)` whose key is `'fixed'`. Notice that this is not the usual 4×4 homogeneous affine matrix we deal with everyday. This dictionary turns out to be an `itkMatrixOffsetTransformBase` which is in LPS convention. In fact, ANTs itself based a lot on Insight Toolkit (ITK)[3], which uses LPS convention.

However, when we play with volumetric data using packages such as NiBabel[4] or spherical data using packages like Vedo[5], PyVista[6], VTK[7], all of a sudden we realize that we are dealing with the RAS coordinate system. Let's say we want apply the affine matrix we just obtained on the reconstructed

[3]https://itk.org/
[4]https://github.com/nipy/nibabel
[5]https://github.com/marcomusy/vedo
[6]https://github.com/pyvista
[7]https://vtk.org/

surface of the same subject (represented by a set of points). It would be very intuitive to want an "true" affine matrix $\tilde{A} \in \mathcal{M}_{4\times 4}$ such that we can move each point $\mathbf{p} \in \mathbb{R}^3$ by just doing the dot product, i.e., $\tilde{A} \cdot \hat{\mathbf{p}}$, where we append a 1 to the last component of each $\mathbf{p}$ to obtain $\hat{\mathbf{p}} \in \mathbb{R}^4$.

At this point, we realize we are facing two problems:

1. Constructing $\tilde{A}$ from the dictionary we read from `affine.mat`.

2. Getting $\tilde{A}$ from LPS to RAS.

3 Solution

The following solution is attributed to frheault[8].

```
import numpy as np
import scipy.io as sio

def ants_mat_to_4x4(ants_mat_fn: str) -> np.ndarray:
    """
    Read in and convert the .mat file from ANTs format to a 4x4
     ↪  matrix.
    """
    # read in .mat file
    _dict = sio.loadmat(ants_mat_fn)
    # define transformation from LPS to RAS
    lps2ras = np.diag([-1, -1, 1])
    # get rotation, translation, and center
    rot = _dict['AffineTransform_float_3_3'][0:9].reshape((3,
     ↪  3))
    trans = _dict['AffineTransform_float_3_3'][9:12]
    center = _dict['fixed']
    # compute offset for \tilde{A}
    r_offset = (- np.dot(rot, center) + center + trans).T * [-1,
     ↪  -1, 1]
    # compute rotation for \tilde{A}
    r_rot = np.dot(np.dot(lps2ras, rot), lps2ras)
    # set offset and rotation
    data = np.eye(4)
    data[0:3, 3] = r_offset
    data[:3, :3] = r_rot
    return data
```

[8]https://github.com/dipy/dipy/discussions/2165

Before explaining in detail how this function works, I will give a brief overview of how an affine transformation is characterized in ITK[9]. The code should be self-explanatory after this introduction.

In ITK, an affine transformation $T : \mathbb{R}^3 \to \mathbb{R}^3$ is given by

$$T(\mathbf{x}) = A(\mathbf{x} - \mathbf{c}) + \mathbf{t} + \mathbf{c},$$

where $A \in \mathcal{M}_{3\times 3}$ is the 3×3 matrix corresponding to the first 9 components of the value of `'AffineTransform_float_3_3'`, $\mathbf{c} \in \mathbb{R}^3$ is the center which can be obtained from the value of `'fixed'`, and $\mathbf{t} \in \mathbb{R}^3$ is the translation which is also the last 3 components of the value of `'AffineTransform_float_3_3'`.

The upper left 3×3 matrix of $\tilde{A}$ would come directly from A. However, remember that we need to apply a transformation, i.e., `lps2ras`, that takes this matrix from LPS to RAS.

The upper right 3×1 column of $\tilde{A}$ would be the "offset". Luckily, the offset is easy to calculate since we are given the formula for it in SimpleITK's documentation[10], i.e.,

$$\mathbf{t} + \mathbf{c} - A\mathbf{c}.$$

Remeber that we still need to apply `lps2ras` to it.

4 Shortcut

There is an easier way of doing this with minimal programming. Suppose ANTs is installed. The following bash code does exactly the same thing but easier.

```
ConvertTransformFile 3 affine.mat affine_hm_ras.mat --hm --ras
```

[9]See https://simpleitk.readthedocs.io/en/master/fundamentalConcepts.html#transforms for more details

[10]https://simpleitk.readthedocs.io/en/master