

ANTs

Advanced Normalization Tools

Script Overview

For an introduction to ANTs, see:

<https://www.neuroinf.jp/fmanager/view/737/bah20150723-alex.pdf>

<https://github.com/stnava/ANTsDoc/raw/master/ants2.pdf>

ANTs is threefold. First, it calculates the transformation needed to register each participant's functional data to the corresponding structural image, which is typically of better resolution. Second, it calculates the transformation needed to register your structural image to a template of your choice (while registration and normalization using fsl's FEAT limits you to using an MNI template), regardless of which space your template is in (whether it is in MNI space or not, whether it is a homemade template or not) and regardless of the difference in resolution between your structural and template images (i.e. can have different voxel sizes and different image dimensions). These calculations use non-linear and linear transformations as well as an additional ANTs-specific fine-tuning algorithm called SyN. Third, ANTs will apply the two previous calculations to normalize each participant's functional data to the template. Thus, each participant's functional data are standardized into a common space to allow data analyses across participants (i.e. the voxel found at a given XYZ coordinate in each participant's data represents the same brain region across participants).

ANTs does so by first calculating the transformation needed to register two images together, with the `antsRegistrationSyN.sh` script that has been written by ANTs creators, and then applying the transforms (functional -> structural & structural -> template) to your functional data using the command `antsApplyTransforms`. The final result will be a new normalized functional data file, leaving your original functional data file intact.

This document gives you a breakdown of 1) what my **ANTs.py** script does, 2) what you need to do yourself so that the script will find & process the files of your choice, and 3) provides highly suggested solutions to consider if the script fails or doesn't do what you want it to do.


1) What Ants .py does:

a. Splits your functional data into single-volume files. For example, if one run (run = block = session) of your task created functional files of 227 volumes (check using the `FSLINFO` or `FSLHD` commands in bash and looking at dim4), splitting your file will result in 227 separate single-volume files! This is done with the command `FSLSPPLIT`. This is due to the fact that you cannot register a multi-volume file (i.e. your 4D functional data) with a single-volume file (i.e. your 3D structural image). The resulting output will be found in your `junk/` sub-folder (see below) under the name `participant#_run#_volume#.nii.gz`.


b. Calculates the transformation needed to register your “moving” file containing the middle volume of your Functional data with your “fixed” Structural image. You don’t need to calculate the transformation between the structural and each of the single-volume functional files you created. Just using the middle volume is enough. It uses the ‘-t a’ transformation, which means the rigid non-linear and the affine linear transformation, since the resolution of functional data is not high enough to use more sophisticated transformation tools. The resulting files are in your `func/` sub-folder (see below) and called `run#_FuncToT1_0GenericAffine.mat` which contains the transformation values of each of the 12 degrees of freedom (you can check it out using MATLAB), another called `run#_FuncToT1_Warped.nii.gz` which shows the resulting brain once the transformation is applied, and `run#_FuncToT1_InverseWarped.nii.gz` which is an inverse visualization of the Warped image (and useless in my script).


c. Calculates the transformation needed to register your “moving” Structural file to your “fixed” Template file. It uses the ‘-t s’ transformation, which is the default and means the rigid non-linear, the affine linear, and the SyN transformation. The outputs are in your `anat/` sub-folder (see below) and are the same as in b. (with the name `FuncToT1` being replaced by `Struc2Temp` and without the prefix `run#`), with an additional `Struc2Temp_1Warp.nii.gz` which shows the deformation needed to get from the moving image to the fixed image with SyN, and `Struc2Temp_1InverseWarp.nii.gz` which is an inverse visualization of the needed deformation image (and useless in my script).


d. Concatenates the two previous transform calculations and apply them to the middle volume of your Functional data, resulting in a `run#_allApplied.nii.gz` file in your `junk/` sub-folder.

 Note that for this step, the reference image is a resampled template with voxel sizes matching the ones of the functional data files. Resampling should be done prior to starting the ANTs script, using the `FLIRT` command.

e. Applies the final transform of `run#_allApplied.nii.gz` file to all single-volume functional data files, one by one, and merges them, one by one, together to form a single normalized final functional data file containing the same amount of volume as your original functional data file (e.g. 227). The result will be `transformed_run#.nii.gz` in your func sub-folder.

 Note that for this step, the reference image is a resampled template with voxel sizes matching the ones of the functional data files.

 The resulting multi-volume 4D functional data file will have the same TR as your template in its file information packet (accessible with the function `FSLINFO` or `FSLHD` under the header `pixdim4`). This does not affect the data but it needs to be fixed in order to match the TR used by your MRI scanner when it required the functional data (e.g. go from `pixdim4 = 1` to `pixdim4 = 2`), as that header will be read by `fsl` when you model as the actual TR used, and modelled as such with no option of changing that in `fsl` itself.

 Your final volume number will appear to be short 1 volume (e.g. 226 instead of 227 when you look at `dim4` using `FSLINFO` or `FSLHD`). This will simply be because the first volume will be considered volume 0 instead of volume 1. As long as you make sure volume 0 is not disregarded when you model your data, and thus that your period of interest during the fMRI run corresponds to the correct volume in your time series, it should be fine.

f. Renames the TR (`pixdim4` in `FSLINFO` or `FSLHD`) using `NIFTI_TOOL`. However, you cannot change just `pixdim4`, but every value in the header, so you should specify voxel and image dimensions as well. To do so, you also need to unzip your final nifti file, and re-zip the new one. The result will be `final_run#.nii.gz` in your func sub-folder.

g. Gives you the code to delete your `junk/` sub-folder and its files, as well as any of the remaining calculation files (which was used between different steps above but will not be used again outside of the context of ANTs). However, it does not run that part of the code unless you uncomment the lines of your choice.

2) Your role in modifying ANTs . py:

a. Get to know where and how your files are organized and modify the script's structure accordingly, if you know python and bash.

OR

a. Set up folders in the following way:

-Have a folder for each participant's data. The folders should be named by the participant's number, ranging from 001 to the total number of participants (e.g. 008 or 024 or 138), with 0's when necessary to keep the folder name consistently 3-digit long. If you want, the participants' folders can have letters before the 3-digit number (e.g. sub_008), but having a 3-digit participant number is super important! You can change line 29 and 30 if that bothers you, but it is not advised.

- Have at least two sub-folders in each participant's folder. One named anat/ that contains the T1 Structural image, that is skull stripped, and one named func/ that contains the functional data files, skull stripped as well.

-Skull stripped functional data files should be named however you want, but the filename should end with a single digit representing your run/block/session number (e.g. functionalfilename1.nii.gz),

-Be aware that each of these two sub-folders will have new output files from the script, which will be left over unless you delete them later using the 'rm' function in bash (looping over all folders with python if possible).

-Have an additional temporary sub-folder in each participant's folder called junk/ that will have all the temporary files, which you can delete later using the 'rm' function in bash (looping over all folders with python if you want).

-Have a folder where your skull stripped template and skull stripped resampled template are.

-Know/Be familiar with your TR, volume number, and general dimensions of all your functional, structural, and template files.

b. Define the variables as presented to you in the **ANTs .py** script after the = sign:

runnum = Write down the number of runs/blocks/sessions, which each have a functional file. No quotation marks!

e.g. runnum = 3

vol = Write down the total number of volumes you have in a functional run. That number should be consistent across all runs and all participants. You can check that by looking at dim4 in FSLINFO or FSLHD in bash. No quotation marks!

e.g. vol = 227

mid = Write down the number of the middle-volume you've selected. It should be vol/2 if you have an even number of volumes or (vol/2)-1 if you have an odd number of volumes. No quotation marks!

e.g. mid = 113

tr = Write down the TR of your original functional data file. You can check this using the FSLINFO command under pixdim4. Use at least one decimal value, with a dot. No quotation marks.

e.g. tr = 2.0

vox_x = Similarly, write down the voxel size on the x axis, found under pixdim1. Use at least one decimal value, with a dot. No quotation marks.

e.g. vox_x = 4.0

vox_y = Similarly, write down the voxel size on the y axis, found under pixdim2. Use at least one decimal value, with a dot. No quotation marks.

e.g. vox_y = 1.7188

vox_z = Similarly, write down the voxel size on the z axis, found under pixdim3. Use at least one decimal value, with a dot. No quotation marks.

e.g. vox_z = 1.7188

path = "Write inside quotation marks the path to where all the participants' folders are.

This should not include the name of any file, just the path. FINISH with a / (don't forget!). Make sure there are no spaces within the quotation marks at all."

e.g. path = "/study4/midusref/DATA/mri/processed/Daphnee/"

It is assumed that the path to a participant's folder will then look like

/study4/midusref/DATA/mri/processed/Daphnee/008/

and that the path to that participant's functional, structural, and junk sub-folders will look like

`/study4/midusref/DATA/mri/processed/Daphnee/008/func/`

and

`/study4/midusref/DATA/mri/processed/Daphnee/008/anat/`

respectively.

`path_length` = Write down NOT in quotation marks the number of / there is in your path.

e.g. `path` = `"/study4/midusref/DATA/mri/processed/Daphnee/"`

e.g. `path_length` = 7

`BOLDfile` = "Write inside quotation marks the name of your functional file for a run, EXCLUDING the run number and EXCLUDING the suffix as well (!), and of course, EXCLUDING the path leading up to that file. DO NOT WRITE a / at the end. Make sure there are no spaces within the quotation marks at all".

e.g. `BOLDfile` = `"functionalfilename"`

This means that a functional data file for your first run is named

`functionalfilename1.nii.gz`

without any other punctuation or space or symbol between the name and the number.

`STRUCfile` = "Write inside quotation marks the name of the skull stripped structural file you want to use for registration, EXCLUDING the path leading up to that file. You should definitely INCLUDE the suffix, which should be `.nii.gz`. This name should be consistent for all participants, and therefore should not mention any participant name, and definitely should not have any run number. DO NOT WRITE a / at the end. Make sure there are no spaces within the quotation marks."

e.g. `STRUCfile` = `"T1_brain.nii.gz"`

This means that your skull stripped structural image file is named

`T1_brain.nii.gz`

for all participants, without any other punctuation or space or symbol or variation.

`path_template` = "Write inside quotation marks the path to your template folder. This should not include the name of any file, just the path. FINISH with a `/` (don't forget!). Make sure there are no spaces within the quotation marks at all."

e.g. `path_t` = `"/study4/midusref/DATA/mri/processed/Daphnee/templatefolder/"`

`template` = "Write inside quotation marks the name of your skull stripped template file, EXCLUDING the path leading up to that file. You should definitely INCLUDE the suffix,

which should be `.nii.gz`. DO NOT WRITE a / at the end. Make sure there are no spaces within the quotation marks.”

e.g. `template = "MAYO_template.nii.gz"`

`template = "Write inside quotation marks the name of your skull stripped template file that has been resample to have the same voxel sizes as the functional data. This template should therefore have a lower resolution. EXCLUDE the path leading up to that file. You should definitely INCLUDE the suffix, which should be .nii.gz. DO NOT WRITE a / at the end. Make sure there are no spaces within the quotation marks."`

e.g. `resampled_template = "MAYO_template_resampled.nii.gz"`

c. Run the script in bash first by running the command `chmod +x ANT5.py` to give yourself the authorization to run that script, followed by the command `./ANT5.py` to run it. This will show you if there are any errors. If things run smoothly, abort the script, and run the script in condor instead, making sure to queue each participant individually.

d. Uncomment and run (preferably, only run them, don't rerun the whole script) the relevant lines at the bottom of the script to delete junk files and files that you will no longer want.

3) Solving potential errors:

In general, don't worry about creating double files if you re-run the script after it crashes. The script is set up so that if an output file already exists for any participant/run, that step is skipped for that participant/run.

-Check what the error in your bash terminal output script or condor logs. The script tells you which step it is in and what it is doing for which participant, so you should have a pretty good idea of where/what the error is. If it is a coding error that is not related to the part you are involved in, that is, where you define your variables, don't fix it unless you know python and bash! Ask someone who is comfortable with both languages to help you 😊

- Check that your set up your variables correct, with or without "", with or without / at the end, and without any space within the "".

- Are all your path set up correctly?

- Do your functional data files finish with the # of the run/sessions? If so, make sure that your BOLDfile includes the part of the name leading up but NOT including the number, and NOT including the suffix .nii.gz.

- Is the suffix of all your functional data files a compressed nifty file (.nii.gz)? It should.

- Check which version of python, bash, fsl, and Ants you are using. It is possible that several updates in each program has created differences in how the code should be written. Check `antsRegistrationSyN.sh -help` and `antsApplyTransforms -help` in bash to see if the code's settings matches what is expected. If not, contact Doug Douglas, Jeanette Mumford, or someone with python coding experience to help.