# HOW TO PERFOM REGISTRATION ON TWO IMAGES

USING ANTS REGISTRATION LIBRARY

GUIDE - DCC LAB

GITHUB AND GROUPE PAGE

*by*

Marc-Antoine Roy
(marc-antoine.roy3@usherbrooke.ca)

Centre de recherche CERVO
May 30, 2024

## Table of content

## Introduction

The following guide as been written to help users register two images together. Registration is the process of mapping an image taken in a moving referential to a certain fixed referential. Suppose you have two images, $I_1$ and $I_2$, of an MRI scan where $I_1$ was taken before a surgical operation and $I_2$ taken after. You goal, you want to see the effect of the surgery compared to the different regions of the brain before the surgery. Since plenty of factors have changed after the surgery, it is most likely that $I_2$ does not directly correspond to $I_1$. The number of planes might not even be the same. Therefore, image registration will map $I_2$ into $I_1$'s referential so that both their planes correspond and the brain will be scaled according to $I_1$'s referential. This guide should contain all the needed information to prepare the software needed to do image registration and explain the different parameters in order to effectively do the image registration. The software used and described here is Advanced Normalization Tools (ANTS) and it's documentation can be found here. While reading this guide, if there are sections or steps that are unclear or unprecise, feel free to message any the author so that the guide can be updated and upgraded.

# 1 Registration with a bash command prompt

The installation process described here is the process for mac users. If your goal is to install it on Windows, you will have to install the Window's Linux Subsystem (WSL) and then follow the steps described in section 1.1. You can find documentation on how to install WSL here. Before installing ANTs, you need other softwares installed on you computer. You need:

— Github installed on your computer. The process to install it is located here.

— Homebrew, to install, you only need to copy this line into a terminal:

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/
  Homebrew/install/HEAD/install.sh)"
```

— Cmake, easy to install with Homebrew, such as:

```
$ brew install cmake
```

— XCode (only for mac users), installed such as:

```
$ xcode-select --install
```

## 1.1 Quick installation process

If you are rushed by the time and you need to do image registration as soon as possible, this section will explain you how to do it. I still recommend that you take a little bit of time to follow the steps in section to know how ANTs is installed and where it is in your computer.

Once you have all the requirements, the quickest way to install ANTs is pretty simple and goes like this:

1. Create a directory on your computer where ANTs will be installed and locate it somewhere you have access on your computer.
2. Click on this link that will forward you to a public github repository.
3. In the master branch, click on the file `installANTs.sh` and download it locally on your computer.
4. Place `installANTs.sh` into the directory you just created.
5. Open a terminal and direct yourself into that directory. To help navigate into the terminal, I recommend this tutorial.
6. Run the command:

   ```
   $ bash  installANTs.sh
   ```
7. The terminal you openend should inform you on the status of ANTs's installation. Normally everything is set once you see the line `Installation complete` in your terminal.

## 1.2 Detailed installation process

Here, we will go through the detailed installation process that the `installANTs.sh` file hides. The process goes like this:

1. Clone the following repository locally on your computer. To do this, still in the terminal, we go to the directory of the github file present on the computer. If this file does not exist, you can create a file that will contain future repositories to be cloned. The command to do so is:

   ```
   $ git clone https://github.com/ANTsX/ANTs.git
   ```
2. Create a directory on your computer where ANTs `install` and ANTs `build` folders will be and locate it somewhere you have access on your computer. Here is an example of an ANTs folder with the name `where_ants_is`. Do one line at a time in the terminal:

   ```
   $ mkdir -p where_ants_is where_ants_is/install
     mkdir -p where_ants_is where_ants_is/build
   ```
3. Move the ANTs folder in `where_ants_is`.
4. It is now necessary to build the modules which will make it possible to download ANTs. To do this, just launch the following two commands:

```
$ cd where_ants_is/build
$ ccmake ../ANTs
```

5. An interface should appear in the terminal. Since this is the first configuration, you must press the c key. Then, we change the variable CMAKE_INSTALL_PREFIX for the path of the file where_ants_is/install that we created. Then, we reconfigure by pressing c to save the changes and we press g to be able to generate the build file.

6. We then return to the terminal and to launch this chosen configuration, we launch:

   ```
   $ make 2>&1 | tee build.log
   ```

   The build.log file will then contain the output of the terminal during the compilation, making it possible to keep the output in memory if there is an error in the compilation. You can also choose to run:

   ```
   $ make -j NBR_THREADS 2>&1 | tee build.log
   ```

   which makes it possible to speed up compilation, because you can choose the number of threads that the computer will use during the operation. Simply replace NBR_THREADS with the desired number of threads.

7. Normally, if everything went well during the build, we should see the message [100%] Built target ANTS in the terminal.

8. Now you need to install ANTs. To do this, we must go to the ANTS-build subfolder, present in the where_ants_is/build folder that we have just compiled. Thereafter, we launch the installation as:

   ```
   $ cd ANTS-build
   $ make install 2>&1 | tee install.log
   ```

   Then, after the execution of these two commands the where_ants_is/install folder should contain the two bin and lib subfolders and the installation is in principle complete.

9. To check if everything is working, you can do the following verification:

   ```
   $ export ANTSPATH=/absolute_path/where_ants_is/install/bin/
   $ export PATH=${ANTSPATH}:$PATH
   ```

   These commands then make it possible to define the ANTSPATH and PATH variables in the environment that we are in as global variables. Be sure to add the abolute path of the where_ants_is folder. For example, /Users/my_user/Documents/where_ants_is, if my /where_ants_is/ folder is located in my documents.

10. We can then launch

    ```
    $ which antsRegistration
    ```

which should show the path of the `antsRegistration` function and:

```
$ bash antsRegistrationSyN.sh
```

which should give the documentation for the function. If it works, then the installation was successful.

## 2 Computing registration on images

### 2.1 Typical registration command

Here, I will go trough a typical ANTs registration function call and I will detail the parameters and their use. It is also important mentionning that in order to specify the number of threads used during ANTs computation, you can define the global variable

```
$ export ITK_GLOBAL_DEFAULT_NUMBER_OF_THREADS=18
```

and specify the number you want. The example shows 18 threads. The function call that will be described here is the following:

```
export ANTSPATH=/absolute_path/where_ants_is/install/bin/
export PATH=${ANTSPATH}:$PATH
export ITK_GLOBAL_DEFAULT_NUMBER_OF_THREADS=18

filename_moving="moving_ref_stack.nrrd"
filename_aligned="aligned_stack_to_ref.nrrd"
filename_fixed="fixed_ref_stack.nrrd"

antsRegistration -d 3 --float 1 -v \
    --output [,$filename_aligned] \
    --interpolation WelchWindowedSinc \
    --use-histogram-matching 0 \
    -r [$filename_fixed, $filename_moving,1] \
    -t rigid[0.1] \
    -m MI[$filename_fixed,$filename_moving,1,32,Regular,0.25] \
    -c [1000x500x250x150,1e-8,10] \
    --shrink-factors 12x8x4x2 \
    --smoothing-sigmas 4x3x2x1vox \
    -t Affine[0.1] \
    -m MI[$filename_fixed,$filename_moving,1,32,Regular,0.25] \
    -c [1000x500x250x150,1e-8,10] \
    --shrink-factors 12x8x4x2 \
    --smoothing-sigmas 4x3x2x1vox \
    -t SyN[0.2,6,0] \
    -m CC[$filename_fixed,$filename_moving,1,2] \
    -c [200x200x200x20,1e-8,10] \
    --shrink-factors 12x8x4x2x1 \
    --smoothing-sigmas 4x3x2x1x0vox \
```

Let's go trough each lines of this function call. First, we exported as global variables `ANTSPATH`, `PATH` and `ITK_GLOBAL_DEFAULT_NUMBER_OF_THREADS` as explained before. We then define the moving referential that we want to map to the fixed referential. We also define the output image, wich will be called `aligned_stack_to_ref.nrrd`. Here, we chose `nrrd` file formats, but in reality the user can use whatever he feels confortable with.

— `-d 3`: The selection of the dimentionnality of the stacks you are working with. For example, if the stacks are zstacks, then you have to specify 3 dimensions, so it will try and match the number of planes for the fixed and moving images. If you only have a single plane, then you can specify a dimension of 2.

— `--float 1`: Enable the computation to use floats and not doubles.

— `-v`: Command to indicate that you want the output to be verbosed, meaning that the ANTs command will executed and the print progression messages to the terminal for the user.

— `--output [,$filename_aligned]`: Simply specify that the image resulting of the computation will be outputed with the name `aligned_stack_to_ref.nrrd`.

— `--interpolation WelchWindowedSinc`: The interpolation type used when computing the registration. There are a few possible choices such as: `Linear`, `Gaussian`, `CosineWindowedSinc`, `WelchWindowedSinc`, etc. The possible types are shown in the `ants.apply_transforms` function from the antspy module here.

— `--use-histogram-matching 0`: Determine if you want the intensities of the fixed and moving images to be matched according to histogram matching algorithms before the registration call. For more information on histogram matching, see this link.

— `-r [$filename_fixed, $filename_moving,1]`: Here, we tell the program to apply an itinial tranform to quickly align the images before registration. The method to chose are 1, match by center of mass or 2, match by point of origin.

— `-t rigid[0.1]`: The first transformation call, where we chose to do a rigid transformation with a gradient step of 0.1. The smaller the gradient step is, the more accurate the computation will be but it may take much longer.

— `-m MI[$filename_fixed,$filename_moving,1,32,Regular,0.25]`: This option is the metric option which measures similarity between the two images. The MI option is based on mutual information of both images and is comparison between the two histograms of the images. In the example, we have 32 bins for MI, and values are sampled regularly in 25% of the voxels. The 1 is the weight for that particular transformation. If we were to do another transformation with a weight of 0.5, they would both contribute equally to the total transform.

— `-c [1000x500x250x150,1e-8,10]`: This option is the convergence levels chosen for the transformation. There will be 4 levels in the whole transform with 4 differents maximum number of iterations: 1000, 500, 250, 150. There is a certain threshold to respect, 1e-8, and if that threshold between each iteration is respected for 10 iterations in a row the computation will pass to the next level.

— `--shrink-factors 12x8x4x2`: It is the division of the resolution at wich the transform is applied for each level. At level 1000, it will have divide the resolution by 12, 500 and 8, 250 and 4 finally 150 and 2. The number of factors much match the number of levels in `-c` argument.

— `--smoothing-sigmas 4x3x2x1vox`: They reprensent the standard-deviation of the smoothing gaussian applied on the images during each levels. They also must match the number of levels. Here we have 1000 and 4, 500 and 3, 250 and 2 finally 150 and 1.

— `-t Affine[0.1]`: Here, we do a second type of transformation, afine. The options are all the same as the rigid transformation, so we will not go trough them again.

— `-t SyN[0.2,6,0]`: Here, we do a third and final type of transformation, SyN, which is a non-linear transformation. The first option is the gradient step again. When computing the gradient step, there are also two other quantities computed, updated gradient field and total gradient field. The second option is the `updateFieldVarianceInVoxelSpace` and makes the update field less sensitive to local deformation. The third option is the `totalFieldVarianceInVoxelSpace` a it is applied on all the deformations computed from the beginning, meaning it makes the image more rigid.

— `-m CC[$filename_fixed,$filename_moving,1,2]`: The metric applied here is based on cross-correlation of the two images. The first option is still the weight and the second is the radius of the metric. The other options following are the same as before and won't be explained.

## 2.2 Tips for registration

When doing your first registration, you can try to call the command with the options above since they are pretty much universal. It may not work as expected and there are a few things you can do to help.

There are a lot of parameters to chose and you may not want to blindly modify some and see the results, because registration can take some time to compute. You can try to run all your transformations separately. SyN transformation and CC metric take a lot of computationnal power and are harder to parametrise. Start by making sure that all of you other transformation seem to work proprely. What I mean by working proprely is that, after the transformation, the resulting image is not completely incorrect. When all the separate transformations work, then try to mash them together to see the result.