8DM20 - Capita Selecta in Medical Image Analysis

Introduction

In the exercises, we use the *elastix* image registration package in *Python* to learn more about nonlinear registration. You need to install elastix and the required Python packages first. The installation instructions are explained in the following. Please make sure to follow all the steps one by one[1].

1. Go this page: https://github.com/SuperElastix/elastix/releases/tag/5.0.0. This page includes the necessary binary files of a stable release of elastix i.e., v5.0.0 on Github.
2. Depending on your operating system (Linux, Windows or Mac), download the corresponding compressed archive from the **Release notes** section.
3. Extract the archive to a folder of you choice. Let us call it `myfolder`.
4. Make sure your operating system can find the program:
   - Windows 7 and Windows 10: Go to the control panel, go to "System", go to "Advanced system settings", click "Environmental variables", add `myfolder` to the variable "path".
   - Linux: Add two new environmental variables to your `.bashrc` file by running the following lines in terminal:

     ```
     - echo "export PATH=myfolder/bin:$PATH" >> ~/.bashrc
     - echo "export LD_LIBRARY_PATH=myfolder/lib:$LD_LIBRARY_PATH" >> ~/.bashrc
     - source ~/.bashrc
     ```

     Note: Fix the `~/.bashrc` path if it is different in your system.
   - Mac: Do the same as Linux, just use `DYLD_LIBRARY_PATH` instead of `LD_LIBRARY_PATH`. Please note that Mac might recognize elastix to be from an unidentified developer. Thus some additional steps are needed which are as follows:
     - Go to `myfolder/bin`. Right click on `elastix` and click on Open. A window appears, which says *elastix cannot be opened because the developer cannot be verified.* Then you should click on Open again. A terminal opens, which you can close afterwards. Repeat the same thing for `transformix`.
     - In the next step choose from Apple menu > System Preferences, click Security + Privacy, then click General. At the bottom of the page you will see *libANNlib-5.0.1.dylib was blocked from use because it is not from an identified developer.* You click on Allow Anyway. In this way you grant an exception for this blocked app.

   Note: If you run into an issue during installation, check here first: https://github.com/SuperElastix/elastix/issues. You might find your answer there. Otherwise, let us know.
5. The next step is to use a Python wrapper for `elastix`, which allows you to call `elastix` and `transformix` from Python. For this step, you can follow the installation instructions from here: https://github.com/tueimage/elastix-py, which are also mentioned in the following briefly:

[1]The instructions are taken from the manual of elastix, Chapter 3, Section 3.2

- Use `pip` package and install the `numpy` and `SimpleITK` by
  `pip install SimpleITK numpy`
- Install from the repository by
  `pip install git+https://github.com/tueimage/elastix-py`

If you have different `pip` installations, use `pip3` instead of `pip` for both steps.

After the installation, the standard approach is to call elastix with two images, the fixed and moving image, and a file containing the parameter settings. The results are saved in a result directory, which needs to be created before running an experiment. In order to avoid overwriting previous outcomes, create a new result directory for each experiment.

In the parameter file you set the specifics for an experiment. There are many parameters to set and sometimes many possible values for them. Most exercises come with a parameter file. If you need an overview of all parameters, what they do and what there values can be, you can find this at http://elastix.lumc.nl/doxygen/modules.html, in particular under Components. The output consists of a registered moving image, *result.x.tiff*. The transformation parameters are saved in a txt file (*TransformParameters.x.txt*) and so are the values of the cost function (*IterationInfo.x.txt*). Finally, there is a log file (*elastix.log*) containing the settings of the experiment, potential warnings and such. A Python script that covers most the functionalities of Elastix is included in the *example.py* file here: https://github.com/tueimage/elastix-py/blob/master/example.py.

Exercises

Sampling

In the folder *example_data* there are two images (*patient1.jpg* and *patient2.jpg*), which contain MR brain data of two different individuals. You can load the images using the `imageio` Python package and the `im = imageio.imread(filename)` functionality. To visualize both the moving and fixed image side by side, you can use the `subplot` functionality of the `matplotlib.pyplot` package which helps you get a better idea of how the images look compared to each other. The images have three equal channels, so you can only take one for visualization `im = im[:,:,0]`.

To understand the registration process, it is sometimes useful to study the search space. This can be done by using the optimizer *FullSearch*, setting a number of values for one of the transformation parameters and evaluating the cost function for those values. See *parameters_samplespace_MR.txt* for an example. In the current setting, it evaluates parameter 0 (rotation around z-axis) from -1.5 to 1.5 in steps of 0.01.

- Run this experiment by calling `el.register` in the regular manner with the two MR brain images and this parameter file. Read the resulting values of the cost function from the *IterationInfo.x.txt* files. You can load the values using the `log = elastix.logfile(Iteration_file_path)` command. Then plot the cost values against the iteration number as `plt.plot(log['itnr'], log['metric'])`, where `plt` is the `matplotlib.pyplot` package (`import matplotlib.pyplot as plt`). You can set a title string to identify each plot e.g. `plt.title('plot title')`. You should see a function with a clear minimum in the centre of the cost curve.

One way of speeding up the registration process is by subsampling, i.e. by not using all voxels in the images to compute the cost function, but a subset. You can use various ways of subsampling by setting the parameter (*ImageSampler*) differently.

- Change it from *Full* to *Grid* (in *parameters_samplespace_MR.txt*, under *Equidistant subsampling*), which performs equidistant sampling on the image grid. It comes with the parameter *SampleGridSpacing* (what do these values mean?). Run `el.register` again and plot the cost function with this equidistant subsampling. What is different with respect to the full sampling?
- Now change *ImageSampler* to *Random*, which includes a parameter for the number of samples. Again run `el.register` and plot the cost function. What is different compared with the previous two plots?
- How do the sampling strategies compare, in terms of effect on the cost function and on computation time? How do you think the strategies will influence the registration success?

B-spline grid spacing

We continue with these images to study the influence of the spacing of the grid points in a B-spline deformation model. You set the spacing with the parameter (*FinalGridSpacingInPhysicalUnits x*). I'll explain the *Final* later. Use parameter file *parameters_bspline_MR.txt.* If you open the file, you can see that the method uses normalized correlation, a standard gradient descent optimization and a single resolution.

- Register patient1 (fixed) and patient2 (moving) five times, using Bspline spacings 64, 32, 16, 8 and 4. View the images and look at the differences in the results (the result image is in *results* directory, in tiff format). How does the spacing influence the outcome? Are any of the results satisfactory?

Multi-resolution registration

Open the file *parameters_bspline_multires_MR.txt.* This file is set to perform a multi-resolution registration. The parameter *FinalGridSpacingInPhysicalUnits* indicates the Bspline spacing at the finest resolution. You can set the spacing manually for the coarser resolutions, but by default the spacing is increased by a factor of 2 at each coarser resolution (as is the case in this file).

- How many resolutions are used? What is the grid spacing at the coarsest resolution?
- Can you detect a difference between this multi-resolution result and the outcomes in the previous single resolution experiments? Can you explain this difference (or absence of it)?

We can see the influence of the multiple resolutions by viewing the search space at the different resolutions of the image pyramid. Open *parameters_samplespace2_MR.txt*. It performs a full search optimization at multiple resolutions using a smoothing pyramid of the images. It will write the search space values to separate IterationInfo files.

- Run `el.register` with the MR brain images and this parameter file. Plot each of the cost functions of the five resolutions using all of *IterationInfo.0.Rx.txt* files and specify the corresponding plot legend for each curve e.g.
  `plt.legend(['Resolution '+str(i) for i in range(5)])`. What differences do you see between these plots? What causes these differences? How does this explain the superior performance of the multi-resolution registration compared with the single resolution registration?

Penalty terms, Jacobian

A deformation modelled with B-splines is often called a free-form deformation, because the B-spline model does not restrict the deformation. The method is free to deform the image in any way (well, almost). In contrast, a deformation model like a biomechanical model restricts the deformation to behave according to the physical properties defined by the biomechanical model. The free nature of B-spline modelling can result in very unrealistic deformations, for instance, containing folding of tissue. A way to resolve is this is to include a penalty term in the cost function.

Read in the fixed and moving images in folder *chest_xrays*. These are in *raw* and *mhd* format, which requires a different reading routine[2]. The *.mhd* files are pure text header files that contain properties of the images. For instance you can read these properties with
`open('chest_ct.mhd') as fp: [print(line) for line in fp]`.

You can also use the `SimpleITK` package to load an image in array format by
`itk_image = sitk.ReadImage('example_data/chest_ct.mhd');`
`image_array =sitk.GetArrayFromImage(itk_image)`, and then visualize it using
`plt.imshow(image_array,cmap='gray'); plt.show()`.

- Register the fixed and moving image using *parameters.txt*. View the resulting image. It should give you the impression of a radiologist high on crack.

The Jacobian determinant of the deformation field gives an indication of local deformation. It can also show erratic deformations like folding. These correspond to negative values. In order to compute the Jacobian determinant of an elastix deformation field, you need to use the `TransformixInterface` class and use the correct path for the input parameters e.g.

```
tr =elastix.TransformixInterface(parameters='results/TransformParameters.0.txt',
transformix_path=TRANSFORMIX_PATH).
```

The `TRANSFORMIX_PATH` points to the installation directory of elastix. Please set accordingly. Try `help(elastix.TransformixInterface)` to get the list of options and definition of the input parameters.

- Compute the Jacobian determinant of the deformation field using `jacobian_determinant`. View the outcome from *results/spatialJacobian.mhd* (it could also be in *.tiff* or *.dcm* format depending on your OS). Then binarize it using level 0 (`out = determinant>0`) and visualize it. Now can you tell whether the folding occurs (negative values occur) and where?

[2]You can find good instructions about working with medical images in Python here: https://github.com/tueimage/essential-skills

As said, a penalty term can regularize the resulting deformations. We add a penalty term called *TransformBendingEnergyPenalty*. It penalizes bending, i.e. the second derivative of the deformation. See *parameterswithpenalty.txt*. The cost function now includes this penalty term and weights to balance the mutual information metric and the penalty term. By varying the weights, you can give more importance to the image metric or to the bending penalizer. Register the images again, this time with *parameterswithpenalty.txt*.

- View the result. How does it differ from the previous result?
- Compute the Jacobian determinant again. Is it correct this time, i.e. are all values positive?
- What is a downside of using a penalty term?