MPHY0030: Programming Foundations for Medical Image Analysis Assessed Coursework 1 (individual tasks) 2022-23

Available on 12th December 2022
Submission before 16:00 – 30th January 2022, on Moodle

Introduction

This the first of two assessed coursework. This coursework accounts for 50% of the module with three independent tasks, and for each task, a *task script* needs to be submitted with other supporting files and data. No separate written report is required.

There are hyperlinks in the documents for further reference. Throughout this document, various parts of the text are highlighted, for example:

Class names are highlighted for those mandatory classes that should be found in your submitted code.
Function names are highlighted for those mandatory functions that should be found in your submitted code.
[5]: square brackets indicate marks, with total marks being 100, for 50% of the module assessment.
"filepath.ext": quotation marks indicate the names of files or folders.
commands: commands run on bash, Python terminals, depending on the context

The aim of the coursework is to develop and assess your ability *a)* to understand the technical and scientific concepts behind the medical image analysis methods, *b)* to research the relevant methodology and implementation details of the topic, and *c)* to develop the numerical algorithms in Python and relevant libraries and packages. Although the assessment does not place emphasis on coding skills and advanced software development techniques, basic programming knowledge will be taken into account, such as the correct use of NumPy arrays, sufficient commenting and consistent code format. Up to [20%] of the relevant marks may be deducted for good programming practice.

Do NOT use this document for any other purposes or share with others. The coursework remains UCL property as teaching materials. You may be risking breaching intellectual property regulations if you publish the details of the coursework or distribute this further.

Python and other packages

No external code (open-source or not) should be used for the purpose of this coursework. No other packages should be used, unless specified and installed within the conda environment below. Individual tasks may have specific requirement, e.g. certain functions should be used for specific implementation. Up to [100%] of the relevant marks may be deducted for using external code. This will be assessed by, on the markers' computers, running the submitted code with a conda environment built from:

```
conda create -n mphy0030-cw python=3.9 pillow=9.2 matplotlib=3.6 numpy=1.23 scipy=1.9 scikit-image=0.19
```

Working directory and task script

Each task should have a task folder, named as "task1", "task2", and "task3". A Python task script should be a file named as "task.py", such that the script can be executed on the bash terminal when the task folder is used as the current/working directory, within the conda environment described above:

```
python task.py
```

It is the individual's responsibilities to make sure the submitted task scripts can be run, in the above-specified conda environment. Even for the data/code available in module tutorials, copies or otherwise automated links need to be provided to ensure a standalone executability of the submitted code. Up to [100%] of the relevant marks may be deducted if no runnable task script is found.

Plotting and visualisation

When the task requires to plot or visualise results, the code should save the results into a PNG file in the respective working directory. Please see examples in the module repository using Pillow or matplotlib, and they may be useful for developing and other visualisation. This should be done programmably in the task script. You can, in addition, submit these png files in the subfolder.

Design your code

The functions/classes/files/questions highlighted (see Introduction) are expected to be found in your submitted code. If not specifically required, you have freedom in designing your own code, for example, data type, variables, functions, scripts, modules, classes and/or extra results for discussion. They will be assessed for correctness but not for design aspects.

Data

The following data are used and click the download link to obtain a local copy for this coursework.

Pelvic MR Volume [download]
The downloaded file "image_train00.npy" contains a volumetric image with an axial in-plane pixel spacing of 0.5 mm/voxel and a slice distance of 2 mm/voxel.

Prostate Segmentation [download]
The downloaded file "label_train00.npy" contains a binary segmentation from the above image, with the same voxel dimensions.

The checklist

This is a list of things that help you to check before submission.

- ✓ The coursework will be submitted as a single "cw1" folder, compressed as a single zip file.
- ✓ Under your "cw1" folder, there should be three subfolders, "task1", "task2", and "task3".
- ✓ The task scripts run without needing any additional files, data or customised paths.
- ✓ All the classes and functions colour-coded in this document can be found in the exact names.
- ✓ Check all the functions/classes have docstring on data type, size and what-it-is for input arguments, outputs and a brief description of its purpose.

Task 1: Constructing and dividing triangulated meshes

- Implement a function surface_dividing, which takes two input arguments 1) a triangulated surface, represented by a list of vertices and a list of triangles, and 2) a scalar value, representing a sagittal plane. This function should return two lists representing two triangulated surfaces separated by the specified sagittal plane. Note: when a plane intersects any triangles, there are two possible cases: a) intersecting with one or two vertices (rarely) and b) intersecting with two edges; if (b), new triangles thus need to be added or formed. [10]
- Implement a task script "task.py", under the folder "task1", completing the following: [15]
  - Load the segmentation file "label_train00.npy" file.
  - Use skimage.measre.marching_cubes algorithm to compute vertex coordinates in mm and triangles for representing the segmentation boundary.
  - Test three sagittal planes of your choice to divide this triangulated surface into two surfaces, left and right. Plot these 6 divided surfaces in a clear and visually comparable manner.
  - Save the visualisation in PNG files in the folder "task1", with informative filenames, such as "case1_left.png" and "case1_right.png".

Task 2: Filtering before resizing

- Implement a class Image3D, which should handle 3D medical images with different voxel dimensions, image sizes and data types. [15]
  - A class constructor __init__ function, which takes a NumPy array representing a 3D image and a tuple of three numerical items for voxel dimension.
  - Consider what a local image coordinate system is defined and implement it with a brief comment describing it. Pre-compute the voxel coordinates in the constructor function.
  - Implement a class member function volume_resize, which takes an input of three-item tuple, which specifies the resize ratio, i.e. the resized image size should be the original image size multiple by this ratio. This function should return an object of the Image3D class. This function should be implemented using scipy.interpolate.interpn.
  - Implement a class member function volume_resize_antialias that takes input arguments 1) a resize ratio and 2) a standard deviation in mm for specifying a Gaussian filter. This function should implement a resizing function with the Gaussian filter applied to the original image before interpolation.
- Compare the two volume resizing images built-in function, by implementing a task script "task.py", under folder "task2", performing the following: [15]
  - Download the "image_train00.npy" file, and use numpy.load to load.
  - Experiment 1 – Volume resizing:
    - Implement three scenarios 1) up-sampling, 2) down-sampling and 3) resampling such that the resized volume has an isotropic voxel dimension.
    - For each scenario, time the two functions, and comment on the difference.
    - For each resized image in Experiment 1, save 5 example axial/transverse slices to PNG files (with clear filenames, e.g. exp1_scenario1_z5_without_filter).
  - Experiment 2 – Investigate aliasing effect:
    - Down-sample the previously up-sampled image (Scenario 1 in Experiment 1) to the original image size.

        - Compute mean and standard deviation of the voxel-level intensity differences between the original image and the down-sampled images (Experiment 2) using the two volume resizing functions, and comment on the differences. Note: you might need to test different resizing ratios and Gaussian standard deviations, in order to see the expected differences.
        - For each resized image, save 5 example axial/transverse slices to PNG files.

Task 3: Warping images by composing rigid transformations

- Implement a class RigidTransform, which should specify a 3D rigid transformation which can warp 3D image volumes. Different from Task 2, all the units in this task should be in voxel, therefore the Image3D class may not be appropriate to use in this task. [15]
    - A class constructor __init__ function, which takes a set of 6 rigid transformation parameters, 3 rotations and 3 translations, as input. Precompute a rotation matrix and a translation vector and stored in the returned class object.
    - Implement a class member function compute_ddf, which takes a three-item tuple, representing the warped image size, and returns a NumPy array that defines a 3D displacement vector (from warped image to original image) at each (warped image) voxel locations. This dense displacement field (DDF) should also be precomputed in __init__ and stored in the returned class object.
    - Describe the image coordinate system you chose to use in the above function comments/docstring. Both the warped image and the original image should follow the consistent coordinate system definition, including origin, orientation and unit (in voxel, for the purpose of this coursework).
    - Implement a class member function warp, which takes a NumPy array, representing a 3D image volume, as input. This function returns a warped image volume in a NumPy array. The intensity values in this returned array should be resampled at the rigidly transformed (specified by the object) new coordinate system, i.e. the warped image.
    - Implement a class member function compose, which takes a second set of rigid transformation parameters as input, and returns a RigidTransform object which represents a rigid transformation that combines the two rigid transformations, i.e. applying the combined transformation should be equivalent to applying them sequentially, the second after the original transformation. This function should update the previously-defined rotation matrix, translation vectors and DDF, in the new returned RigidTransform object to represent the composed transformation.
- Implement a task script "task.py", under folder "part3", test the composed transformation. [15]
    - Load the image file "image_train00.npy".
    - Manually define the ranges of translation and rotation parameters, according to the coordinate system you defined in RigidTransform class. Comment on your rationale, such that the resulting warped images are reasonably visible in this coursework.
    - Experiment 1 – test the implemented image warping and transformation composing:
        - Randomly sample 3 sets of rigid transformation parameters T_1, T_2 and T_3 from above-defined range.

    - Instantiate 3 objects such that they represent three rigid transformations, T_1, $T_1 \oplus T_2$ and $T_1 \oplus T_2 \oplus T_3$, where $\oplus$ represents transformation composing and the latter two of which are composed transformations.
    - Compare the two warped images, between 1) using the composed transformations and 2) by applying T_2 and T_3 sequentially on previously warped images.
    - For each warped image volume (you should have 5 now), save 5 example axial/transverse slices to PNG files. Comment on your finding in the task script.
- Now add a flag `flag_composing_ddf` in the RigidTransform class, such that, when the flag is `True`, the function compose uses a different algorithm to compute the composed DDF. This new algorithm calls a new function composing_ddfs, to take two DDFs as input and returns the composed DDF, i.e. without using composed rotation matrix and translation vector. [15]
  - Experiment 2 - in the same task script "task.py":
    - Repeat Experiment 1 after enabling `flag_composing_ddf=True`.
    - Compute the voxel-level difference between each of the two warped images. This is the difference between those with and without using composing_ddfs. Report the mean and standard deviation of the intensity differences.
    - For the two newly warped image volumes, using the saved 5 example axial/transverse slices in PNG files. Comment on the visual comparison to those obtained from Experiment 1.