MPHY0030: Programming Foundations for Medical Image Analysis Assessed Coursework 1 2021-22

Available on 19th November 2020
Submission before 23:59 – 17th 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 or MATLAB terminals, given 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. only NumPy can be used for certain function implementation. Up to [100%] of the relevant marks may be deducted for using external code. This will be assessed by, on the markers' computers, the submitted code will be tested with a Python 3.7 environment built from:

```
conda create -n mphy0030-cw python=3.7 pillow=8.4 numpy=1.21 scipy=1.7 scikit-image=0.18
```

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, for compatibility with those environments that do not support graphics, such as WSL or remote setups. Please see examples in the module repository using Pillow. N.B. matplotlib and plotting functions in scikit-image cannot be used in the task scripts, although it may be useful for developing and visualisation. Up to [20%] of the relevant marks maybe deducted if this is not followed.

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, you should have 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 Distance Transform in NumPy

- Implement a function distance_transform_np, which takes a 3D volumetric binary image as input and returns its 3D Euclidean distance transform. The function should accept the second argument as the voxel dimensions in each axis and the computed distance transform should be in the unit of millimetre. You should use *only* NumPy for implementing this function. [6]
- Briefly describe the algorithm you used in the function docstring. [3]
- Compare the built-in function distance_transform_edt in scipy.ndimage with your implementation. Implement a task script "task.py", under folder "task1", performing the following:
    - Download the "label_train00.npy" file, and use numpy.load to load. [1]
    - Compute distance transform of the *segmentation boundary* using the two implementations, i.e. distance_transform_np and distance_transform_edt. [4]
    - Time the speed of two implementations, and comment on the difference. [3]
    - Compute the mean and standard deviation of the voxel-level difference between the two implementations, and comment on the difference. [3]
    - Save 5 example slices to PNG files (filename being slice index), across the volume, together with their corresponding distance transform results for each of the two algorithms. [5]

Task 2 Triangulated Surface and Normal Vectors

- Implement a function surface_normals_np, which takes a triangulated surface, represented by a list of vertices and a list of triangles, as input, and returns two types of normal vectors 1) at vertices and 2) at triangle centres. You should use *only* NumPy for implementing this function. [3]
- Implement a task script "task.py", under folder "task2", completing the following:
    - Load the segmentation file "label_train00.npy" file. [1]
    - Use measre.marching_cubes algorithm to compute vertex coordinates in mm, triangles and vertex normal vectors, for the surface that represents the boundary of the segmentation. [3]
    - Determine a reasonable metric for comparing normal vectors and use it to compare the vertex normal vectors computed from the two implementations, measre.marching_cubes and surface_normals_np , comment on the difference. [3]
    - Design a method to compare the vertex normal and the triangle-centre normal vectors, computed from surface_normals_np, and use it to compare their difference and comment on the results. [4]
    - Use a 3D Gaussian filter with a scalar standard deviation σ to smooth the binary segmentation. Then compute the three normal vectors from the two implementations on the smoothed segmentation. Test a *reasonable range* of σ values, and comment on the impact of σ. [5]
- Implement a visualisation script "vis.py", under folder "task2", for:
    - Visualise the surface triangulation together with the two types of normal vectors, in a clear and visually comparable manner. [5]
    - Save the visualisation in PNG files in folder "task2" (with informative filenames). [1]

Task 3 Image Affine Transformer

- Implement a class Image3D, which should handle 3D medical images with different voxel dimensions, image sizes and data types.
    - A class constructor function, which takes a NumPy array representing a 3D image.

        - Consider what the precomputing can be done during construction of a new image object, and implement them with comments. [3]
        - Consider what the best way to specify local image coordinates, and implement them with brief comments explain the rationale. [4]
    - A 3D image warping function, warp, which takes an extra input object from class AffineTransform, see below, and computes a warped 3D image, with all voxel intensities interpolated by trilinear interpolation method. [5]
- Implement a class AffineTransform, which has the following member functions.
    - A class constructor function, which takes a vector of transformation parameters as input and completing the following computations:
        - Check the length of the input vector, allowing 6/7 or 12 degrees of freedom (DoFs), for respective rigid or affine transformation types. This constructor should also allow `None` as input for random affine generation. Raise error for all other cases. [3]
        - Precompute the transformation matrix in homogeneous coordinates, using rigid_transform or affine_transform, and save it in a class member variable. [2]
        - In the case of random transformation, call random_transform_generator.
    - A random affine transform generating function random_transform_generator, which returns a random affine matrix in homogeneous coordinates, considering:
        - Design and implement a *reasonable method* for generating random affine transformation of a 3D image with a customisable scalar parameter to control the *strength* of the transformation. [5]
        - Briefly explain your method and its rationale in the function docstring.
    - Implement the rigid_transform. (for DoF=6/7) and affine_transform (for DoF=12) for taking the transformation parameter to return the transformation matrix in homogeneous coordinates. [6]
- Implement a task script "task.py", under folder "part3", performing the following:
    - Load the image file "image_train00.npy" and instantiate an Image3D object. [1]
    - Manually define 10 rigid and affine transformations, demonstrating a variety of rotation, translation, scaling, general affine and combinations of them. [3]
    - Generate the warped images using these transformations. [3]
    - Generate 10 different randomly warped images and plot 5 image slices for each transformed image at different z depths. [5]
    - Change the strength parameter in random_transform_generator. Generate images with 5 different values for the strength parameter. Visualise the randomly transformed images. [4]
- Implement a visualisation script "vis.py", under folder "task3", for:
    - Visualise these transformations from both of those manually-defined and randomly-generated, using a sparsely-sampled image grid in 3D in mm. [5]
    - Save the visualisation in PNG files in folder "task2" (with informative filenames). [1]