Scaphoid Project Documentation

Short explanation

This project creates a 3D segmentation and analysis for the Scaphoid bone and the fracture inside of the bone. More information regarding the project and explanation of the algorithms can be found in the project's report.

Flow

The general flow of the program is to create a segmentation and then create an analysis. Since the segmentation has room for improvement, this was separated into 2 sections.

When running GUI.py, which is the main logic, the main page of the program opens, where you can select if you want to choose segmentation or analysis. In both processes, the first input is an output folder, second input is the original CT scan in a nifti format. The last input is seeds for the bone and the fracture (for the segmentation process), or fixed segmentation of the bone and fracture, with seeds for the Radius bone and 2 seeds in the Capitate bone (for the analysis process).

Color choosing

When creating the seeds, the colors are important. Use color 1 (red) for Scaphoid Bone, color 2 (green) for the fracture, color 3 (blue) for the Radius bone and 2 colors for the Capitate bone: Color 4 (yellow) for the top marking, Color 5 (light blue) for the bottom marking.

Compilation

The tool is compiled into an EXE file using GUI.spec file and pyinstaller. To compile, write in the **terminal** the line 'pyinstaller GUi.spec'. The compilation will run and create a folder called build, which contains the exe file.

Code Files

Comparison_to_GT.py:

A helper file to compare between the segmentation and the ground truth using dice coefficient.

Bone.py:

The parent class for all of the bones, can be extended to other bones. Most of the functions are basic and will be relevant to most bones, excluding get_neighbors and region_growing.

Radius.py:

A class for the Radius bone, exists to overwrite the region_growing of the bone due to the radius special structure.

Scaphoid.py:

An extension of the Bone class. Since the main goal of the project is to work on this bone and it's fracture, this class have functions that relate to the fracture as well.

region_growing for the bone was overwritten, as well as a new function for region growing of the fracture and get_neighbors that are relevant for the fracture. The Analysis process can be easily expanded, see next section.

GUI.py:

The main logic of the program. Opening the GUI allows the use to select which process he wants to run (segmentation or analysis). For each process there is a function that has the entire logic of the process (and each expansion can be added there).

For future development

All of the functions are documented inside of the code. Future development can be expanding the amount and type of geometrical features, better UI, increasing the accuracy of the segmentation and even expanding the algorithm to other bones.

UI notes:

Changes to the UI are welcome, and can easily improve the experience.

In GUI.py the function 'Segmentation process' has the logic of the segmentation process:

1. extracting seeds from the bone and fracture
2. region growing to create a segmentation for the bone
3. region growing to create a segmentation of the fracture.
4. save all the files (files name are constants in GUI.py).

In GUI.py the function 'analysis process' has the logic of the analysis process:

1. loading the scaphoid bone and fracture from the file
2. getting seeds for the radius and capitate bones.
3. creating a segmentation for the radius bone
4. extracting PCA from the radius bone
5. divide the fracture and bone into quarters based on the PCA.
6. Get all the geometrical features from the fracture
7. Get the angle between the capitate and the radius bone
8. Save all the files.

Note that if you wish to create a better UI, the flow of the segmentation should stay the same, while for the analysis process you need to make sure you extract the PCA before the division and the angle.

Segmentation accuracy notes:

For each bone there is a different Region growing algorithm, based on the same logic.

In Scaphoid.py the function region_growin_from_input creates the segmentation for the bone. The documentation shows each part of the algorithm. The neighbor choice and threshold is the easiest to change and will take change in the segmentation.

In Scaphoid.py the function segment_fracture_region_growing_mean creates the segmentation for the fracture. In this algorithm, changing the function

neighbors_for_fracture will change the segmentation. Different functions regarding the thresholding might improve the segmentation results.

In Radius.py the function region_growing_from_input creates the segmentation for the bone. Because of the attributes of the bone (hollow in the middle and thick at the edges) the algorithm first fills all of the hollow voxels, and then the thicker ones.

The segmentation for the bones (both Scaphoid and Radius) achieve good results. The main improvement can be achieved by increasing the accuracy of the fracture segmentation. In order to check the results, use the file comparison_to_GT.py. In this file you'll find the following functions to compare between the algorithm's segmentation and the ground truth:

1. compare: gets the algorithm's segmentation path, the ground truth path, color of the bone (1 by our design) and the color of the fracture (2 by our design). The function prints the dice coefficient for the bone, and for the fracture.
2. compare_every_slice: gets the algorithm's segmentation path, the ground truth path, color of the bone (1 by our design) and the color of the fracture (2 by our design). The function creates a file called '#Scan_results.csv' and writes into it the dice coefficient of the bone and the fracture in each slice. This function might be more informative for research.

Geometrical features notes:

Most of the geometrical features regarding the fracture are in the Scaphoid.py file. The only geometrical feature in GUI.py is the angle between the capitate bone and the radius bone, inside of the analysis_process function.

In Scaphoid.py the function get_geometric_features returns a dictionary, where the key is the name of the feature, and the value is a string of the value of the feature. The reason the keys and values are strings, is to export them into a txt file with the measurements. You can create many more functions that create more geometrical features, and simply add them in the dictionary format, in order to increase the number of features.

Flatness features is not fully complete and without a lot of research (due to lack of information from the medical team). It can contain a more precise value.

You can also create a function for geometrical features in different bone, or regarding the scan itself, just make sure you add it into the 'geo_features' dictionary in GUI.py in 'analysis_process' function. The format here as well should be the name of the feature as key (string), and the value with measurements as value (string as well).

Expansion to other bones:

You can expand the Bone class to multiple different bone. Main functions that are helpful for time saving (such as getters, save and load) will exists. Your main goal will probably be to create a new segmentation, so you should overwrite the region_growing_from_input function, or the _get_neighbors function.

You can also run the default region_growing_from_input function in order to get a general idea regarding the changes that needs to be done to improve the segmentation.

Visualizing results:

We recommend application ITK-snap in order to visualize the results. With this app you can convert the original CT scan (DICOM format) into nifti format, which is easy to work with using python. You can load a scan and load the segmentation on it in order to view the results in the of the segmentation over the scan in a convenient way.