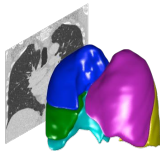
Pulmonary Toolkit 

https://github.com/tomdoel/pulmonarytoolkit

Tutorial 3

Programming with the Pulmonary Toolkit

Version 1.3

Tom Doel

Overview

In this tutorial we introduce the Application Programming Interface (API) which allows you to use results from the Toolkit in your own programs. We illustrate this with examples showing how to obtain, visualise and save results.

The tutorial code is in the repository: `Scripts/PTKTutorial.m`

Topic covered

- Architecture and starting the Toolkit using the API
- Running algorithms and getting results
- Viewing images with the PTKViewer
- Saving out image results
- Saving out airway results

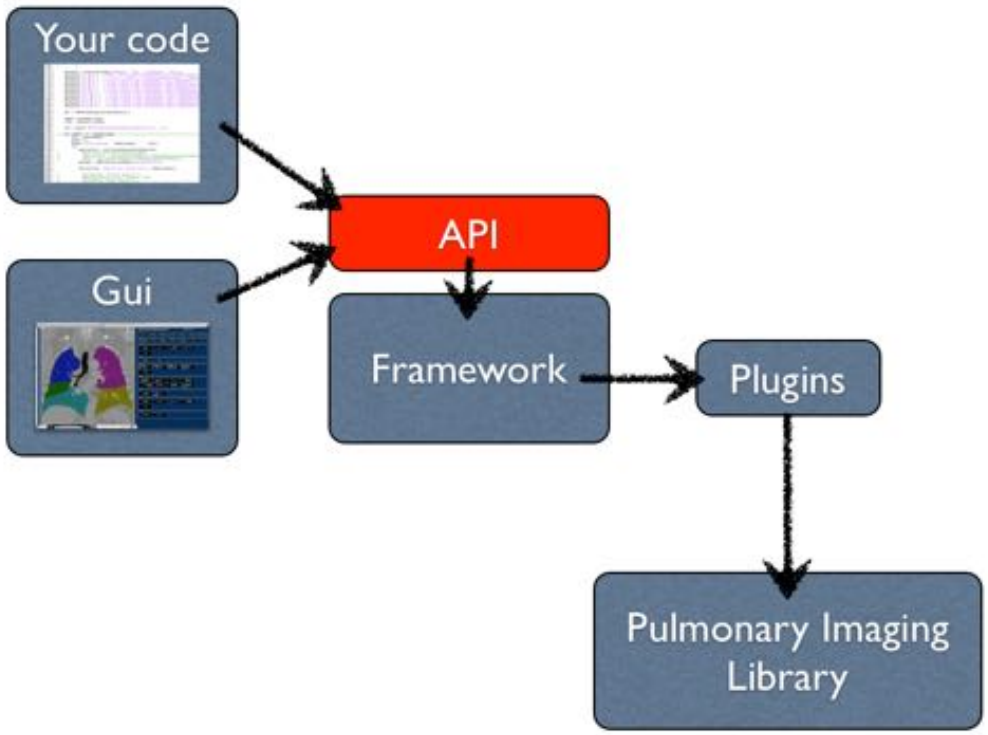
Requirements

You should follow the documents “Installing the Pulmonary Toolkit” and Tutorials 1 & 2. A basic understanding of object-orientated programming in Matlab is helpful.

1. Architecture

The Toolkit provides you with a series of Matlab methods which you can call from your own code. Using standard jargon, we call these functions the **application programming interface** (API). The API provides a simple way for you to run algorithms and fetch results while hiding away most of the Toolkit's complexity.

The graphical user interface (GUI) is built using these API interfaces, so you can use the GUI code as example code for how to use the API. A simplified illustration of the Toolkit architecture is shown below.



Analysis algorithms are represented by class files called **Plugins**. This tutorial will focus on how to run these plugins and obtain the results. In later tutorials we will create new plugins to add new features.

You may also wish to use functions in the **Pulmonary Imaging Library**, which contain lower-level routines for image analysis, saving and visualising data.

2. Starting the Toolkit using the API

For this tutorial, start up Matlab and switch to the Pulmonary Toolkit directory. You can work in the Command Window or create a script if you prefer.

The first thing we need to do is set up Matlab's paths so that the Toolkit can find all its required files. Run the following:

```
>> PTKAddPaths;
```

Next, we create an object of the `PTKMain` class.

```
>> ptk_main = PTKMain();
```

I'm not going to go into the details of object-oriented programming here, but what this line does is to create a new object of class `PTKMain`, and provides access to it through a handle variable `ptk_main`. When the new object is created, its constructor function performs some initialisation. The handle variable `ptk_main` gives us access to the functions we need.

Getting a PTKDataset for the images you want to work with

The Pulmonary Toolkit allows you to work with many datasets simultaneously. Each dataset is accessed using a separate `PTKDataset` object.

To get a PTKDataset object for a particular dataset, we will use the `ptk_main` object we created above. Each volume (series) is accessed by using methods in `ptk_main` to create a new object of class `PTKDataset` which represents that image volume (series). The simplest is the helper method `Load()`

```
>>source_path = 'put-the-path-to-your-dataset-here';
>>dataset = ptk_main.Load(source_path);
```

Now `dataset` is a handle variable to the `PTKDataset` object for this dataset. You can create different handle variables for different datasets. This allows you to work with multiple datasets simultaneously.

The `Load()` method will import all data from the given directory and return to you a single `PTKDataset` object for the first series found. If you want more control over the files that are imported, you can instead use the `CreateDatasetFromInfo` method.

`CreateDatasetFromInfo` requires you to specify the list of files in a special `PTKFileInfo` object. Specifying this manually would be quite tedious if our dataset comprises 500 files. However, there is a helper method which will find all Dicom files in a particular directory.

```
>>file_infos = PTKDicomUtilities.GetListOfDicomFiles(source_path);
```

This will store a list of filenames in a `PTKFileInfo` object. We can use this to obtain a `PTKDataset` object for this dataset as follows:

```
>> dataset = ptk_main.CreateDatasetFromInfo(file_infos);
```

3. Creating a PTKDataset using a UID

The above two methods `Load()` and `CreateDatasetFromInfo()` are slow because they first import the data into the PTK's internal database before creating the `PTKDataset` object. However, this import step only needs to be performed once, as PTK will remember your imported data, even between Matlab sessions.

You can speed up your code considerably by using a different method, `CreateDatasetFromUid()`, to create a `PTKDataset` object without forcing the data to re-import. `CreateDatasetFromUid()` is faster because it assumes the data are already imported. It needs to know the **UID** (unique identifier) for the dataset to load.

The PTK gives every dataset has a unique identifier. We can use this identifier to obtain a `PTKDataset` object for our data, instead of specifying every file as we did above.

For Dicom files, this UID is a tag in the Dicom file called the **Series Instance UID**. It is guaranteed to be the same for all images in the series, but different for any other images. One advantage of using this UID is that it will be the same for anyone using the same set of data.

However, the UID itself does not give the Toolkit any indication of where the files are. Hence, you can only use this method if the data has previously been imported into the Toolkit.

You could do this for example by loading up the graphical user interface (GUI) and then importing or loading the data. Alternatively, you can use another PTKMain method called `ImportData()`:

```
>> uid = ptk_main.ImportData(source_path)
```

You only need to import the data once; once the data has been imported, the Toolkit knows how to link the UID to the image files.

How do you get the UID? The above function will return it in the variable **uid**. Or, if you load up the GUI, you can copy the UID to the command window and the clipboard using the **Copy UID** button. For Dicom files, you could also examine the metadata to find the Series Instance UID tag

Once you have imported the data and found the UID, you can create a PTKDataset using the command

```
>> dataset = ptk_main.CreateDatasetFromUid(uid);
```

4. Running algorithms and getting results

The dataset object you have obtained allows you to run algorithms and fetch results for the data encapsulated in that `PTKDataset` object.

The key method is `GetResult`:

```
>> lobes = dataset.GetResult('PTKLobes');
```

This obtains the result of the algorithm `PTKLobes`, which is the main lobe segmentation algorithm. If the algorithm has previously been run, the result is fetched from the disk cache. Otherwise the algorithm is run.

The type of result you get will depend on the algorithm. For the lobes, this returns a handle to a `PTKImage` object. This is an image matrix with additional metadata (such as voxel size, etc).

> *Note: if you want to access the underlying image matrix, it's easy to do this by accessing the* ***`RawImage`*** *property, i.e.* ***`lobes.RawImage`****.*

Not all results are images. For example, compute the airway centreline:

```
>> airway_centreline = dataset.GetResult('PTKAirwayCentreline')
```

This returns a structure with a variety of properties. The `AirwayCentrelineTree` property is a connected tree structure containing the points in the airway centreline.

5. Documentation

PTK functions and classes are documented using Matlab's help system. To show help for a class or function, use `doc`, for example:

```
>> doc PTKImage
```

to bring up a help window, or `help` to show documentation in the command window:

```
>> help PTKImage
```

6. Viewing images with the PTKViewer

To view the lobe image, you can use a little viewer application:

```
>> PTKViewer(lobes);
```

This opens a viewing window similar to the one used in the mainGUI. The controls are the same. You will need to scroll though the window before you actually see anything, because you are just looking at the lobe segmentation and it is blank at the image edges.

> *Note: the viewer window in PTKViewer is built on a Matlab panel, making it easy to add to your own graphical applications in you wish.*

We are looking at the lobe segmentation, but how do we see the original data? Another plugin called **PTKLungROI** will fetch us the CT data (cropped down to the lung ROI or "region of interest"). We can bring up another window to view this:

```
>> ct_image = dataset.GetResult('PTKLungROI');
>> image_viewer = PTKViewer(ct_image);
```

If you can't see the image you may need to adjust the window and level (see tutorial 1). You can do this with the mouse or sliders, or using the following commands:

```
>> image_viewer.ViewerPanelHandle.Window = 1600;
>> image_viewer.ViewerPanelHandle.Level = -600;
```

How do we superimpose the lobes on the CT image? The following command will do this by setting the `OverlayImage` property of the `PTKViewerPanel` object:

```
>> image_viewer.ViewerPanelHandle.OverlayImage = lobes;
```

> *Note that the **`PTKViewer`** is really only a wrapper. The main functionality is provided by a **`PTKViewerPanel`** object, which is accessed through the **`ViewerPanelHandle`** property.*

7. Viewing images in 3D

3D visualisation is provided by the function `PTKVisualiseIn3D`. You can vary the amount of smoothing

```
>> smoothing_size_mm = 4;
>> PTKVisualiseIn3D([], lobes, smoothing_size_mm, false);
```


8. Saving out image results

Now let's save out the images. There are a variety of helper functions for this. To save out the 3D imaging data use the PTKSaveAs method to choose the filename and file type:

```
>> PTKSaveAs(lobes, 'Patient Name', '~/Desktop');
```

The above command saves out the entire 3D dataset at the original resolution. A dialog lets you choose the file format. If you want to explicitly save in a particular format, you can call method such as

```
>> MimSaveAsNifti(lobes, '~/Desktop', 'lobes.nii');
```

What about the fusion image of the CT and lobe mask? The viewer has a `Capture()` method which generates a screenshot of the visible viewing area. Assuming the viewer from section 6 is still open, you can capture a screenshot:

```
>> frame = image_viewer.ViewerPanelHandle.Capture();
```

You can write this out as a bitmap file using the Matlab function `imwrite`.

```
>> imwrite(frame.cdata, '~/Desktop/lung_and_lobes.tif');
```

> *Unlike the PTKSaveAs function, the screen capture will depend on the resolution of your image as displayed on the screen. So resize the window to get a better image.*

PTK also provides a function for printing out multiple screen captures from a 3D viewer onto a single figure.

```
>> PTKShow2DSlicesInOneFigure(image_viewer.ViewerPanelHandle,
PTKImageOrientation.Coronal, 20);
```

Note the coronal orientation, and 20 is the number of slices skipped between each image. Decrease to show more images.

9. Saving out airway results

Earlier, we obtained the result of the airway centreline algorithm using the command:

```
>>airway_centreline = dataset.GetResult('PTKAirwayCentreline')
```

We can export this centreline to several formats. Let's save to VTK:

```
>> PTKSaveTreeAsVTK(airway_centreline.AirwayCentrelineTree,
'~/Desktop', 'MyTreeFilename',
PTKCoordinateSystem.DicomUntranslated, ct_image);
```

As with other PTK functions, the Matlab help provides information about the function and its parameters:

```
>> doc PTKSaveTreeAsVTK
```

Coordinates are saved out in mm, but you need to tell the function which coordinate system to use, since the origin is not always consistent between applications (see Tutorial 2). You do this through the fourth parameter, which is an enumeration of type `PTKCoordinateSystem`. Note that Dicom coordinates depend on the image used to generate them, and so you need to provide an image in the final parameter so PTK can correctly work out the coordinates. Any `PTKImage` image will do as long as it's from the same dataset.

> *Note: PTK coordinates are image-independent, so if you specify the cooridnate system* ***`PTKImageCoordinateSystem.PTK`****, then you don't need to provide an image argument.*

