dicomtools Documentation

Release 0.0.1

Steven Engler

Jul 17, 2017

Contents

The dicomtools package is designed to simplify many common DICOM processing steps, and is based on the `pydicom` package. It's designed primarily for research use. This code is still very basic and the structure will likely change. It tries to somewhat follow the DICOM standard, or to at least avoid vendor-specific attributes.

Contents:

CHAPTER 1

Examples

These examples demonstrate some of the basic use-cases of dicomtools.

Reading DICOM Data

You can either read a single DICOM file, multiple DICOM files from the same series, or a DICOMDIR file. A *dicomtools.series.DicomSeries* object will be returned, except in the case of *dicomtools.dicom_read.read_dicomdir()*, which will return a list of *dicomtools.series.DicomSeries* objects.

Reading a DICOMDIR file:

```
>>> import dicomtools
>>> dicomtools.dicom_read.read_dicomdir('data/DICOMDIR')
[<dicomtools.series.DicomSeries object at 0x00000000054A9EB8>,
 <dicomtools.series.DicomSeries object at 0x0000000007EF3710>,
 <dicomtools.series.DicomSeries object at 0x0000000007EF3DD8>]
```

Reading series of DICOM files:

```
>>> import dicomtools
>>> dicomtools.read_dicom_series(['data/im_001.dcm', 'data/im_002.dcm', 'data/im_003.
↪dcm'])
<dicomtools.series.DicomSeries object at 0x0000000008EF85C0>
```

Reading a single DICOM file:

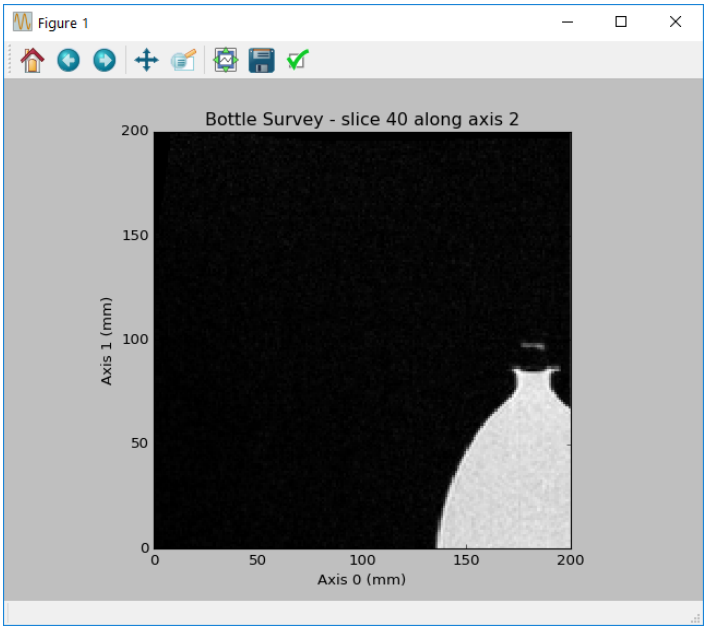
```
>>> import dicomtools
>>> dicomtools.read_dicom('data/im_dat.dcm')
<dicomtools.series.DicomSeries object at 0x0000000006718128>
```

Building a 3D Volume Dataset

If a *dicomtools.series.DicomSeries* object represents 3D data as a multi-frame dataset or multiple DICOM files within a series, you can easily process this using the *dicomtools.volume.DicomVolume*.

View a single slice of the 3D multi-frame dataset:

```
>>> import dicomtools
>>> series = dicomtools.read_dicom('data/im_dat.dcm')
>>> volume = dicomtools.volume.DicomVolume(series)
>>> dicomtools.visualization.plot_slice(volume, 2, 40)
```



Moving Between Image/Pixel and DICOM Patient Coordinates

The *dicomtools.coordinates* module contains useful functions for transforming points between coordinate systems, but with a *dicomtools.volume.DicomVolume* object, it's much simpler.

Transform pixel `(5,5)` to the position in patient coordinates:

```
>>> volume = dicomtools.volume.DicomVolume(series)
>>> img2pat = volume.build_image_to_patient_matrix()
>>> img2pat
array([[   0.   ,    0.   ,   -1.2  ,   59.4  ],
       [   1.136,    0.   ,    0.   , -180.   ],
       [   0.   ,   -1.136,    0.   ,  180.   ],
       [   0.   ,    0.   ,    0.   ,    1.   ]])
>>> pixel_position = [5, 5]
>>> dicomtools.coordinates.transform_vectors(img2pat, pixel_position)
array([  59.4  , -174.318,  174.318])
```

Transform pixel `(4,5,6)` to the position in patient coordinates:

```
>>> volume = dicomtools.volume.DicomVolume(series)
>>> img2pat = volume.build_image_to_patient_matrix()
```

```
>>> img2pat
array([[   0.   ,    0.   ,   -1.2  ,   59.4  ],
       [   1.136,    0.   ,    0.   , -180.   ],
       [   0.   ,   -1.136,    0.   ,  180.   ],
       [   0.   ,    0.   ,    0.   ,    1.   ]])
>>> pixel_position = [4, 5, 6]
>>> dicomtools.coordinates.transform_vectors(img2pat, pixel_position)
array([  52.2  , -175.455,  174.318])
```

Transform a position in patient coordinates back to a pixel position:

```
>>> import numpy as np
>>> volume = dicomtools.volume.DicomVolume(series)
>>> img2pat = volume.build_image_to_patient_matrix()
>>> pat2img = np.linalg.inv(img2pat)
>>> pat2img
array([[   0.   ,    0.88 ,    0.   ,  158.4  ],
       [  -0.   ,   -0.   ,   -0.88 ,  158.4  ],
       [  -0.833,   -0.   ,   -0.   ,   49.5  ],
       [   0.   ,    0.   ,    0.   ,    1.   ]])
>>> dicomtools.coordinates.transform_vectors(pat2img, [59.4, -174.318, 174.318])
array([ 5.,  5.,  0.])
>>> dicomtools.coordinates.transform_vectors(pat2img, [52.2, -175.455, 174.318])
array([ 4.,  5.,  6.])
```

Transform multiple positions simultaneously (faster than transforming each individually):

```
>>> volume = dicomtools.volume.DicomVolume(series)
>>> img2pat = volume.build_image_to_patient_matrix()
>>> img2pat
array([[   0.   ,    0.   ,   -1.2  ,   59.4  ],
       [   1.136,    0.   ,    0.   , -180.   ],
       [   0.   ,   -1.136,    0.   ,  180.   ],
       [   0.   ,    0.   ,    0.   ,    1.   ]])
>>> pixel_positions = [[4,5,6], [5,5,0]]
>>> dicomtools.coordinates.transform_vectors(img2pat, pixel_positions)
array([[  52.2  , -175.455,  174.318],
       [  59.4  , -174.318,  174.318]])
```

Exporting DICOM Images to PNG

The *dicomtools.export* module contains useful functions for exporting images.

Exporting a *dicomtools.volume.DicomVolume* pixel data to multiple image files:

```
>>> volume = dicomtools.volume.DicomVolume(series)
>>> volume.export_images('images_dir', 'image')
```

Exporting a single image slice:

```
>>> volume = dicomtools.volume.DicomVolume(series)
>>> im_data = volume.info['pixel_data'][:,:,10]
>>> dicomtools.export.export_image_to_png(im_data, 'images_dir/image.png')
```

CHAPTER 2

Library Reference

dicomtools package

Submodules

dicomtools.coordinates module

dicomtools.coordinates.**build_image_to_patient_matrix**(*origin_position*, *pixel_spacing*, *row_vec*, *column_vec*, *slice_vec=None*)

Get a matrix to transform a pixel coordinate to a DICOM patient coordinate. The pixel coordinate corresponds to the center of that pixel/voxel.

The DICOM standard defines the patient coordinate system as:

- x -> increasing to the left hand side of the patient
- y -> increasing to the posterior side of the patient
- z -> increasing toward the head of the patient

source: https://public.kitware.com/IGSTKWIKI/index.php/DICOM_data_orientation

dicomtools.coordinates.**build_patient_to_physical_matrix**(*patient_position*)

This function builds a rotation matrix to transform a position in patient coordinates to a position in physical coordinates. This physical coordinate space is arbitrary, but it simply undoes the effect of the patient position. The transformation matrix is the identity matrix for a patient in feet-first prone position. This is useful if you have a robot within the imaging device, and want a consistent coordinate system regardless of the patient position. For example, the up direction will always remain the same for both prone and supine positions. Due to the unknown orientation of the imaging device, we cannot assign direction labels to these physical axes, so you must test it yourself.

Possible patient positions are:

- HFP = head first-prone

- HFS = head first-supine
- HFDR = head first-decibitus right
- HFDL = head first-decubiturs left
- FFP = feet first-prone
- FFS = feet first-supine
- FFDR = feet first-decibitus right
- FFDL = feet first-decibitus left

source: https://public.kitware.com/IGSTKWIKI/index.php/DICOM_data_orientation

dicomtools.coordinates.**build_translation_matrix**(*translation*)

For a given translation vector, this function builds the corresponding transformation matrix.

Example:

```
>>> build_translation_matrix([4,5,6])
array([[1 0 0 4]
       [0 1 0 5]
       [0 0 1 6]
       [0 0 0 1]])
```

dicomtools.coordinates.**expand_transformation_dimension**(*transformation*, *new_size*, *move_translation*)

This function takes an $n \times n$ transformation matrix and expands it to a $new\_size \times new\_size$ matrix. It expands the array, fills in ones along the diagonal, and moves the translation values to the new right column if requested.

Example:

```
>>> test = np.array([[1,2,3],[4,5,6],[0,0,1]])
>>> test
array([[1 2 3]
       [4 5 6]
       [0 0 1]])

>>> expand_transformation_dimension(test, 6, True)
array([[1 2 0 0 0 3]
       [4 5 0 0 0 6]
       [0 0 1 0 0 0]
       [0 0 0 1 0 0]
       [0 0 0 0 1 0]
       [0 0 0 0 0 1]])
```

dicomtools.coordinates.**transform_vectors**(*transformation_matrix*, *vectors*)

This function applies (LHS) the $n \times n$ transformation matrix to a list of vectors and returns a list of the transformed vectors.

The vectors can be any length less than n. The vectors will be zero-filled so that they have length $n - 1$. If the input is one-dimensional, it is assumed that a single vector was given and a one-dimensional vector will be returned.

dicomtools.dicom_read module

dicomtools.dicom_read.**read_dicom**(*dicom_file*)

Build a DicomSeries object containing the DICOM.

dicomtools.dicom_read.**read_dicom_series**(*dicom_file_list*)
Build a DicomSeries object containing all of the given DICOMs in a series.

dicomtools.dicom_read.**read_dicomdir**(*dicomdir_file*)
Build a list of DicomSeries objects, one for each series in the 'DICOMDIR' file.

dicomtools.export module

dicomtools.export.**export_image_to_png**(*image*, *filename*)
Given a 2-dimensional image, save it to a file.

dicomtools.export.**export_stack_to_png**(*images*, *axis*, *directory*, *filename_prefix*)
Given a 3-dimensional array, save each slice to a file. Use the 'axis' argument to describe which axis to iterate over.

dicomtools.series module

class dicomtools.series.**DicomSeries**(*dicom_list*)
Bases: object

A class for storing and processing DICOM series. Instances are verified to all have consistent SeriesInstanceUID attributes.

get_instances_with_image_data()
Get all of the DICOM instances in the series which contain image data (has the pixel_array property).

get_instances_without_image_data()
Get all of the DICOM instances in the series which do not contain image data (do not have the pixel_array property).

dicomtools.visualization module

dicomtools.visualization.**plot_slice**(*dicom_volume*, *slice_axis*, *slice_index*, *figure=None*)
Plot a single slice (along axis 0, 1, or 2) of a *dicomtools.volume.DicomVolume* using matplotlib.

dicomtools.volume module

class dicomtools.volume.**DicomVolume**(*dicom_series*)
Bases: object

Simplifies working with 3D DICOM data.

build_image_to_patient_matrix()
Get a matrix to transform a pixel coordinate to a DICOM patient coordinate. The pixel coordinate corresponds to the center of that pixel/voxel.

The DICOM standard defines the patient coordinate system as:

- x -> increasing to the left hand side of the patient
- y -> increasing to the posterior side of the patient
- z -> increasing toward the head of the patient

source: https://public.kitware.com/IGSTKWIKI/index.php/DICOM_data_orientation

export_images(*directory*, *filename_prefix*, *axis=2*)
Save slices of the volume to images. The pixels in the resulting images will be square, regardless of the DICOM pixel size. These images should not be expected to have perfect pixel-accuracy, and compression may be used.

get_dimensions_in_mm()
Get the dimensions in millimeters for each axis of the volume. Returns a list of length 3.

For 3 pixels...

```
+---+   +---+   +---+
| 1 |   | 2 |   | 3 |
+---+   +---+   +---+

|-|-------|-------|-|
 ^     ^       ^     ^
 |     |       |     |
 |     |       |     ---- 1/2 pixel_size
 |     |       --------- 1 pixel_spacing
 |     ----------------- 1 pixel_spacing
 ----------------------- 1/2 pixel_size
```

dicomtools.volume.**compare_volume_metadata**(*volume1*, *volume2*)
This compares the volume metadata (position, pixel_size, etc) and shape of the pixel data, but not the actual pixel data. Returns True if equal, otherwise returns False.

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index