PlatiPy - Getting Started

Robert Finnegan & Phillip Chlap

August 2020

Installation

Terminal

```
$ pip install -r requirements.txt

or

$ conda install --yes --file requirements.txt
```

requirements.txt

```
numpy
pyyaml
nibabel
itk >= 5.0a1
vtk
SimpleITK
scikit-image
click
pydicom
loguru
```

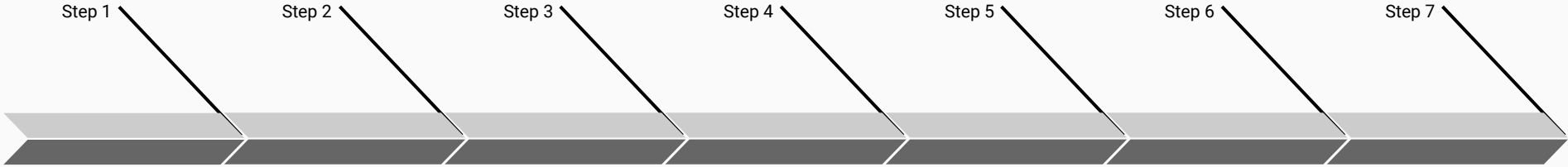
Getting started with PlatiPy is easy!

All software is written in Python (version 3), and to install the required packages we recommend using Pip or Conda.

Check out the manuscript below for some general information on this framework.

Finnegan, et al (2019). Feasibility of multi-atlas cardiac segmentation from thoracic planning CT in a probabilistic framework. *Physics in Medicine & Biology* 64(8) 085006. Link.

Overview of the segmentation pipeline



(Automatic Cropping)

Optional. Reduces the image size by cropping based on the bounding box containing aligned atlases.

Initial Registration

Co-registration of atlases using one of the available transforms: *rigid, affine, similarity, translation.*

DIR

Deformable co-registration. Currently, the fast symmetric forces demons algorithm is the default, though B-splines is available.

(Iterative Atlas Selection)

Optional. This step can remove poorly registered atlases automatically.

Label Fusion

If multiple atlases are used, the labels have to be combined (or *fused*) to generate a probabilistic segmentation map.

(Vessel Splining)

Optional. This step can generate consistent segmentations of vessels using an automatic splining procedure.

Output Labels

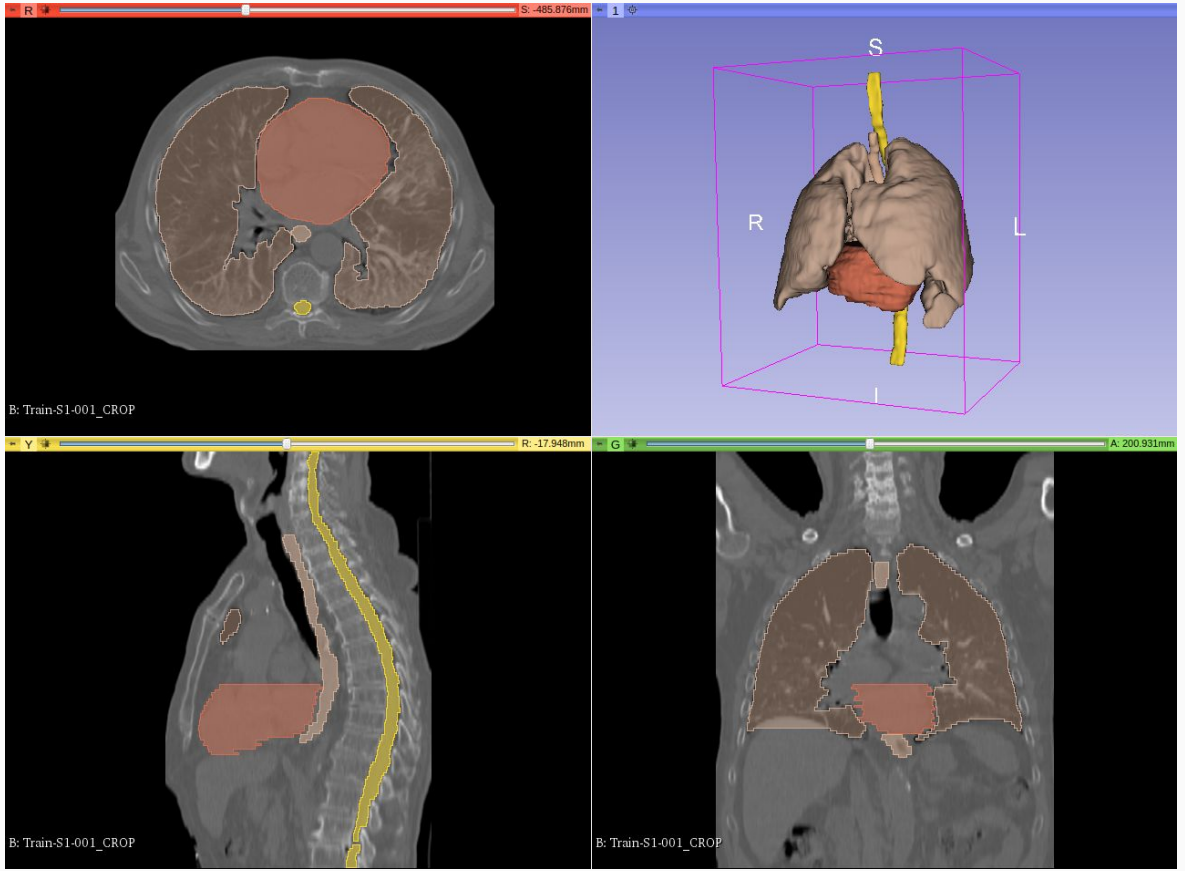
The automatic segmentations are generated, pasted back into the original image space, and saved.

Running an example

We have made a Jupyter notebook containing an example for end-to-end segmentation. Check it out (in **segmentation/examples/**):

cardiac_segmentation.ipynb

There are many options available, which are generally separated by the step of the pipeline they are used for. These will be described throughout this document.



An example from the atlas set, showing the delineated structures.

Step 0. Setting the atlas options

Relevant Options

```
"atlasSettings": {
      "atlasIdList": ["001","002","003","004","005","006","007"],
      "atlasStructures": ["Heart","Esophagus","Lung_L","Lung_R","SpinalCord"],
      "atlasPath": "../tests/data/dynamic/lung",
      "atlasImageFormat": "LCTSC-Train-S1-{0}/CT.nii.gz",
      "atlasLabelFormat": "LCTSC-Train-S1-{0}/Struct_{1}.nii.gz",
     "autoCropAtlas": True
}
```

These settings provide the information required to read in the atlases, and define which structures will be included in the automatic segmentation.

In **atlasImageFormat** and **atlasLabelFormat** the brace characters represent the atlas identifier - *{0}*, and the structure name - *{1}*.

There is one additional option, **autoCropAtlas**, which will crop each atlas image to the smallest bounding box containing all of the selected structures. This can significantly improve the efficiency of image registration and is recommended.

Step 1. Automatic Cropping

In addition to cropping the atlas images, it is also possible to crop the target image. However, since there are no labels, an alternative method must be used.

We provide such a method, which quickly aligns a few atlases. After this, the target image is cropped to the bounding box defined by the union of these atlas images.

The **expansion** defines an optional extension to the bounding box, given as the [axial, sagittal, coronal] extension in mm (in each direction).

Relevant Options

```
"autoCropSettings": {"expansion": [0,0,0]}
```

Step 2. Initial Registration

Relevant Options

```
"intialRegSettings": {
    "initialReg": "Similarity",
    "options": {
        "shrinkFactors": [16, 8, 4],
        "smoothSigmas": [0, 0, 0],
        "samplingRate": 0.75,
        "defaultValue": -1024,
        "optimiser": "gradient_descent",
        "numberOfIterations": 50,
        "finalInterp": sitk.sitkBSpline,
        "metric": "mean_squares",
    },
    "trace": False,
    "guideStructure": False
```

The first stage of atlas registration is the initial registration.

The **initialReg** is the transformation type: *Rigid, Affine, Similarity, Translation* are the currently available options.

A multi-resolution staging can be configured:

- **shrinkFactor:** relative voxel scaling used in each resolution stage
- **smoothSigmas:** standard deviation of 3D Gaussian filter used to smooth images (in mm)

There are several **optimiser** options: *gradient_descent, gradient_descent_line_search, LBFGSB*.

The **finalInterp** can be one of *sitk.sitkNearestNeighbor, sitk.sitkLinear, sitk.sitkBSpline* which defines how the moving image is resampled in the target image space.

The boolean flag **trace** will print information to the terminal (only use for debugging, can cause memory leaks).

guideStructure will evaluate the metric within a masked region, define this using the (string) name of the atlas label (e.g. “Heart”).

Step 3. Deformable Image Registration

The next stage of atlas registration is deformable image registration.

A multi-resolution staging can be configured. This behaves differently depending on whether **isotropicResample** is used.

If **isotropicResample** is set to *True*, **resolutionStaging** will set the voxel size (in mm) at each stage.

If **isotropicResample** is set to *False*, **resolutionStaging** sets the shrink factor (similar to that in Step 2).

Again, **trace** should only be used for debugging since it can cause memory leaks.

As before, **smoothingSigmas** defines the standard deviation of the Gaussian filter used to smooth the images at each resolution stage (in mm). If set to zero, no smoothing will be performed.

The **ncores** defines the number of CPU cores allocated to DIR.

Relevant Options

```
"deformableSettings": {
    "isotropicResample": True,
    "resolutionStaging": [16, 8, 2],
    "iterationStaging": [15, 15, 15],
    "smoothingSigmas": [0, 0, 0],
    "ncores": 8,
    "trace": False,
}
```

Step 4. Iterative Atlas Removal

Relevant Options

```
"IARSettings": {
    "referenceStructure": "Heart",
    "smoothDistanceMaps": True,
    "smoothSigma": 1,
    "zScoreStatistic": "MAD",
    "outlierMethod": "IQR",
    "outlierFactor": 1.5,
    "minBestAtlases": 5,
    "project_on_sphere": False,
}
```

Finnegan et al. (2020) Localised delineation uncertainty for iterative atlas selection in automatic cardiac segmentation. *Physics in Medicine & Biology* 65(3) 035011. Link.

Errors in registration occur. The IAR procedure was designed to automatically identify these, and remove atlases where there may be errors. It is based on the segmentations of a particular **referenceStructure**.

This procedure generates a model of surface variations at each point, described with a **zScoreStatistic** for which there are two options:

- *MAD*: *z = (score - median)/(median absolute deviation)*
- *STD*: *z = (score - mean)/(standard deviation)*

The MAD procedure is more robust.

For each atlas, the surface variations are aggregated to a *Q-value*, with higher values corresponding to more overall variation. Any outliers, which may represent errors, are removed. These are determined with the **outlierMethod** and **outlierFactor**:

- *IQR*: Q > (third quartile) + outlierFactor * (inter-quartile range)
- *STD*: Q > (mean) + outlierFactor * (standard deviation)

Q-value statistics can be calculated using a subset of atlases, which is given by **minBestAtlases**.

This procedure can be used by projecting surface variation onto a sphere (using **project_on_sphere**), which is more robust but only suitable for convex surfaces. Using this projection also allows Gaussian smooth (if **smoothDistanceMap** is *True*) with filter with scale **smoothSigma**.

Step 5. Label Fusion

Once each atlas is registered, the propagated structure labels have to be combined. This produces a probability map, from which the final segmentation is derived using a threshold between 0 and 1.

This process is called label fusion. Each type of label fusion has different parameters (**voteParams**):

- *unweighted*: each atlas has equal contribution, no parameters necessary
- *global*: each atlas contribution is weighted by the sum of squared differences in image intensities between the atlas and target images. No parameters necessary.
- *local*: for each atlas, the weight is spatially varied depending on the squared difference in intensities at each voxel. Requires two parameters, *sigma* - the scale (in mm) of a Gaussian kernel used to smooth the weight map, and *epsilon* - a small number required to avoid division by zero.
- *block*: similar to *local*, except the weight is calculated from a small block around each voxel. Requires two parameters, *blockSize* - the size of the block in voxels (e.g. 3 → 3x3x3 block), and *gain* -

If no parameters are set, the default values will be used. The only other required value is the probability threshold, which is set for each structure (**optimalThreshold**).

Relevant Options

```
"labelFusionSettings": {
    "voteType": "unweighted",
    "voteParams": {},
    "optimalThreshold":    {"Heart":0.5,
                            "Lung-Left":0.5,
                            "Lung-Right":0.5,
                            "Esophagus":0.5,
                            }
}
```

Step 6. Vessel Splining

For some structures, an alternative to label fusion is a splining procedure. This was designed for delineating coronary arteries, but here we give a demonstration using the spinal cord.

The **splineDirection** can be one of *x* (left-right) , *y* (post-ant), or *z* (sup-inf).

The splining procedure can be stopped with the **stopCondition** based on either a minimum number of atlases (*count*) or minimum area in mm^2 (*area*), which will be defined in the **stopConditionValue**.

The settings for the vessel radius, spline direction, and stopping conditions are defined using a dictionary, and so can be different for each structure for which this procedure is applied.

Relevant Options

```
    "vesselSplineSettings": {
        "vesselNameList": ['Spinal-Cord'],
        "vesselRadius_mm": {'Spinal-Cord': 6},
        "splineDirection": {'Spinal-Cord': 'z'},
        "stopCondition": {'Spinal-Cord': 'count'},
        "stopConditionValue": {'Spinal-Cord': 1}
}
```

Step 7. Final Output

Finally, the automatic segmentation is pasted back into the original image space. No additional options are needed for this step.

The automatic segmentations can then written to NifTI images. This can be changed to other formats.

We provide some code to generate a figure of orthogonal image slices with contours of the automatically segmented structures, as shown on the right.

After this, the automatic segmentations can be processed further, used for analysis, or imported into other software.

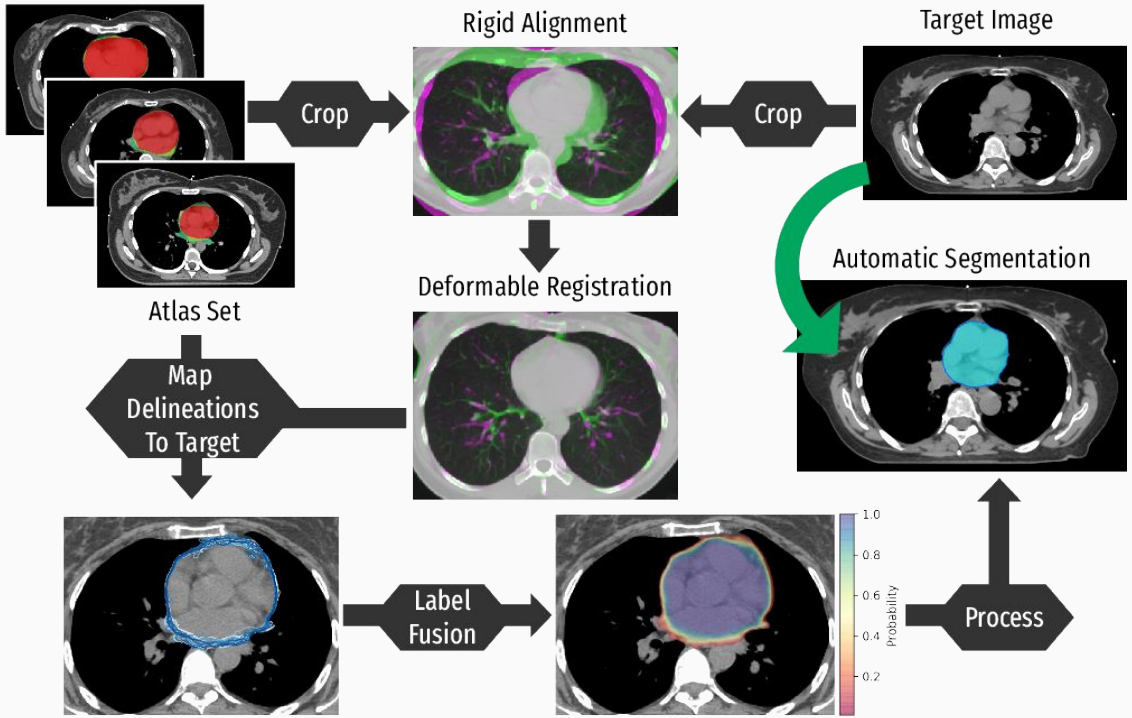
What next?

This software is designed as a library, rather than an application.

The different processes used in the segmentation pipeline can be modified, adapted, swapped out, or iterated in different orders.

A simple way to start to play around is to check out the code in **segmentation/cardiac/run.py** and investigate how the entire pipeline is built up from the different functions provided by the PlatiPy framework.

After that, we would encourage users to check out other parts of the framework (e.g. the registration code) to see what else is available and how it might be used.



Overview of the atlas-based framework. The modular design means the entire pipeline is flexible and can be adapted to individual problems.