User Manual

Introduction

See Workshop paper

Requirements

The following tools and packages are required for the software:

External tools:

Chimera	1.11.2

Python packages:

python	3.6
matplotlib	2.0.2
numpy	1.13.1
scikit-learn	0.19.0
scipy	0.19.1

Installation

Installation of chimera

Go to https://www.cgl.ucsf.edu/chimera/download.html and follow the installation guide for the release you wish to download. Be sure to grab a release at least as new as version 1.11.2 as there is no guarantee that the software will function properly with earlier releases.

Installation of python

The recommended way is to download the latest Anaconda release (4.4.0 as of writing this document) from https://www.anaconda.com/download/ . Using Anaconda the default environment will include all necessary packages for the software, but it will have to be executed using the correct python executable. The python executable should be present in the installation directory (commonly C:\ProgramData\Anaconda3 in windows).

The packages can also be installed individually by following the installation instruction for each package for your desired platform.

After you have installed all the requirements make sure that all the relevant paths are set appropriately. These are:

In `config.txt`:

CHIMERA_PATH – This variable should be set to full path to the chimera executable on your machine.

CHIMERA_UTILS_PATH – This variable should be set to the full path to the directory of the ChimeraUtils within the software.

See the section about the configuration file for more information about it.

Workflow

Outline

The workflow consists of six steps that need to be done. The steps are as follows:

1. Generating a set of templates
2. Generating a set of training tomograms
3. Training an SVM on the templates and tomograms generated
4. Generating or downloading tomograms that will be evaluated
5. Evaluating the tomograms using the trained SVM
6. Collecting and viewing the results

There are three ways to execute these steps that offer decreasing degree of flexibility. These are

1. Python API
2. Shell API
3. FlowTest.py

All three use a file named config.txt as the configuration file. The configuration file must be present in the directory from which the code runs.

Python API

1. Generate templates:

   In the three dimensional case, since the templates are already saved, generating new templates might not be necessary, in which case the templates only need to be loaded.

   Generating templates is handled by a number of methods, all of which are present in TeplateGenerator.py:

```
generate_tilted_templates_2d(angle_res)
generate_templates_3d(output_path, angle_res, templates_type)
load_templates_3d(templates_path)
```

   The parameters in all three methods are as follows:

   `angle_res` – Angular resolution to be used when generating the templates, i.e. the size of the jump, in degrees, between two angles in every direction relevant (phi for two dimensions, phi, theta and psi for three dimensions)

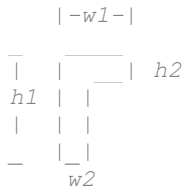
   `template_type` – is the type of the templates to be generated, and can be any one of `GEOMETRIC_3D`, `PDBS_3D` and `ALL_3D`. This variable indicates whether to use geometric shapes, templates downloaded from PDBS or both.

`output_path/templates_path` - is the FULL path to the directory within which the resulting templates will be created and save, or loaded from in the case of the third function.

The return value of all three methods is an iterable of template tilts that allows for subscripting, this could be a list of lists (e.g. the third item in the second value is the third tilt of the second template).
Three dimensional templates are generated using Chimera as a subprocess and therefore must be saved. Regardless, the second method loads and returns the generated templates after they were saved.

geometric.txt and pdbs.txt

It is possible to define which templates are to be generated by Chimera when generating the templates.
To change the geometric templates that are created the `geometric.txt` file should be edited.
Each line in the file corresponds to a template that will be generated. There are three possible geometric shapes: sphere, cube, L-shape (named L in the file).
After the vertical line parameters that define the geometric shapes are specified.
The parameters of the shapes are as follows:
cube| [edge of the cube]
sphere|[radius of the sphere]
L|[w1, w2, h1, h2, zdepth] such that



And zdepth is the size of the shape in the z-direction
To change the pdb templates, all that is required is to add the path to the `.pdb` (in respect to ChimeraUtils directory) file that you wish to generate a template from to the `pdbs.txt` file. The files should be in the form of the path to them. The parameter after the vertical line is used by Chimera to decide on the resolution with which to generate the templates. This in turn defines the size of the tomogram that will be generated. As a general rule, the smaller the parameter the better is its resolution and the bigger the template will be.
Example: Say that you wish to add a template from the file `1k4c.pdb` with resolution of 15 and the file is within the ChimeraUtils directory. We will add the following line to the `pdbs.txt` file:

```
.\1k4c.pdb|15
```

If we wish get new `.pdb` files to make into templates then we need to go to a data bank that supplies `.pdb` (e.g. www.rcsb.org, www.ebi.ac.uk/pdbe/emdb) and download the files from there.

2. Generate tomograms.
   The methods for generating tomograms are the same for both step (2) and step (4) while the third method is used not to generate tomograms for training but rather after evaluation, in the creation of the result tomogram. The methods are present within TomogramGenerator.py:

```
generate_random_tomogram(templates, criteria, dim, noise=False)
generate_random_tomogram_set(templates, criteria, number_of_tomograms, dim, seed=None,
noise=False)
generate_tomogram_with_given_candidates(templates, composition, dim)
```

   The parameters to all three methods are as follows:
   `templates` – Templates as are loaded or generated in the previous step.
   `criteria` – A list of the specifications for which templates to use. This list should have the structure of list of integers where an integer in the `i`'th position is the number of copies of the `i`'th template, e.g. `[2 3]`. The different copies of a template are likely to be of different tilts.
   `number_of_tomograms` – The second method, unlike the other two, is a generalization of the first such that it returns a python generator[1] that generates a number of tomograms using the first function. The number of tomograms to be generated is set using this parameter.
   `dim` – The dimension of the tomograms to generate. This value should be the same as the dimension of the templates that are used, i.e. 2 if the templates are two dimensional.
   `seed` – Seed to be specified for the generation of the tomograms.
   `noise` – Boolean value indicating whether to use noise in the generation of the tomograms or not.

   The return values of the first and third methods are a Tomogram object while the return value of the second method is a generator of Tomogram objects.

3. Training an SVM on the resulting templates and tomograms.
   Creating and training an SVM is achieved via the following method, which is present in SvmTrain.py:

```
svm_train(templates, tomograms, test_list=None)
```

   This method uses the templates and training tomograms, as generated in previous steps, and trains an SVM with them.
   `templates` – The templates to train the SVM with, as generated or loaded in previous steps.
   `training_tomograms` – The tomograms on which the SVM will be trained, as generated or loaded in previous steps.
   `test_list` – This is a variable for debugging the function and can be left as None.

   The return value of the methods has the following structure:

[1] See https://wiki.python.org/moin/Generators for more information about python generators. Not to be confused with the methods in our code that called generators.

```
(svm, (EulerAngle.Tilts, templates))
```

A Tuple containing the trained SVM and as its second value another Tuple containing the templates as its own second value. `EulerAngle.Tilts` is information about the tilts that are used by the templates. This tilts item is necessary when the results are to be saved and reloaded because the templates themselves do not contain this information. To reload it from file, set the field `EulerAngle.Tilts` to its corresponding value within the result.
The reason for this is to keep the SVM and its templates coupled together, as an SVM is specific to the templates it was trained on.

4. Evaluating some tomograms using the trained SVM.
   Before this step can be executed, some tomograms that will be evaluated must be present. Creating these tomograms can be done in the same way as step (2).
   The evaluation of a tomogram or a set of tomograms is achieved via the following method, which is present in SvmEval.py:

   ```
   svm_eval(svm_and_templates, tomograms, test_list=None)
   ```

   This function uses an SVM, as returned from `svm_train`, to evaluate the tomograms, which must be a list, even if containing a single tomogram.
   `svm_and_templates` – An SVM with which the tomograms will be evaluated.
   `evaluation_tomograms` – A list of the tomograms that are to be evaluated.
   `test_list` – This is a variable for debugging the function and can be left as None.

   The return value of the function is a list of lists of candidates for all of the tomograms in the input, such that item 0 in the output is a list of candidates for tomogram 0 in the input.
   The resulting tomogram can be reconstructed using a candidates list. Candidates list functions as a composition and given the corresponding templates can be used in the third function for tomogram generation, `generate_tomogram_with_given_candidates`, to reconstruct the tomogram.

5. Viewing the results of the evaluated tomograms.
   The results can be viewed using the following method which is present in SvmView.py:

   ```
   svm_view(evaluation_tomograms, templates, output_candidates)
   ```

   This method prints a summary of the result of the evaluation of the first tomogram only.
   `evaluation_tomograms` - The set of tomograms that were evaluated in the previous step.
   `templates` – The templates with which the tomograms where evaluated.
   `output_candidates` – The output of the previous step.
   Notice that both the output and the input of the previous step are used in this one, apart for the SVM itself.

   The function does not return anything.

Example

See examples/python_flow_example

Shell API

In all the shell commands, `python` must be the distribution containing all the relevant packages (e.g. Anaconda3\python.exe) and `Main.py` is the path from the current working directory to the Main.py script. Then comes the subcommand which can be either one of `{generate templates, generate tomograms, train, eval, view_results}`
Positional arguments must be given in the order they are defined. It is recommended to supply all the positional arguments before any optional ones. Optional parameters might be required depending on the positional arguments.

1. Generate templates.
   The usage documentation for the template generation shell command is as follows:

```
usage: Main.py generate templates [-r ANGULAR_RESOLUTION]
                                  [-t {GEOMETRIC_3D,PDBS_3D,ALL_3D}]
                                  {CHIMERA} out_path

positional arguments:
  {CHIMERA}
                        The kind of generator to use when creating the
                        templates
  out_path              Path to save in the generated template

optional arguments:
  -r ANGULAR_RESOLUTION, --angular_resolution ANGULAR_RESOLUTION
                        Angular resolution to use in the generation of the
                        templates, i.e. the size of the jump in the angles in
                        every relevant direction
  -t {GEOMETRIC_3D,PDBS_3D,ALL_3D}, --template_type {GEOMETRIC_3D,PDBS_3D,ALL_3D}
                        Type of templates to be generated
```

   The `Chimera` generator requires that both `-r` and `-t` will be supplied and only works for three dimensional templates.
   A possible run of this command is as follows:

```
$> python Main.py generate templates CHIMERA C:\Chimera\Templates\ -r 60 -t PDBS_3D
```

   This command will generate templates using Chimera and store them under the out_path, C:\Chimera\Templates. In the generation of the templates an angular resolution of 60 degrees will be used and the templates use the PDBS_3D type.

   The format of the templates is the same as the format used in the python API.

2. Generate tomograms.
   The usage documentation for the tomogram generation shell command is as follows:

```
usage: Main.py generate tomograms [-c CRITERIA [CRITERIA ...]]
                                  [-n NUM_TOMOGRAMS]
                                  {RANDOM} template_paths out_path
                                  out_path

positional arguments:
  {RANDOM}              The kind of generator to use when creating the
                        tomograms
  template_paths        Path to the templates to be used in the generation
  out_path              Path to save in the generated tomograms

optional arguments:
  -c CRITERIA [CRITERIA ...], --criteria CRITERIA [CRITERIA ...]
                        Criteria to be used by the tomogram generator
  -n NUM_TOMOGRAMS, --num_tomograms NUM_TOMOGRAMS
                        Number of tomograms to create
```

The `RANDOM` generator requires that both `-c` and `-n` will be supplied.
The criteria, specified following `-c`, are turned into an array of integers and used in the same way as the criteria in the python API.
A possible run of this command is as follows:

```
$> python Main.py generate tomograms RANDOM C:\Chimera\Templates\
C:\Chimera\Templates\tomograms.pkl -c 2 3 -n 1
```

This command will generate a list of tomograms using the `RANDOM` generator and the templates from C:\Chimera\Templates\. The criteria used in the generation will be [2, 3] and the number of tomograms generated this way will be 1.
The result of the generation will be stored as a pickle in the file C:\Chimera\Templates\tomograms.pkl.

3. Training an SVM.
   The usage documentation for the training shell command is as follows:

```
usage: Main.py train -t templatepath -d datapath svm

positional arguments:
  svm                   Path to save in the created svm.

optional arguments:
  -t templatepath, --templatepath templatepath
                        Path to the templates to be trained with.
  -d datapath, --datapath datapath
                        Path to the tomograms to be trained on.
```

A possible run of this command is as follows:

```
$> python Main.py train my_svm.pkl -t C:\Chimera\Templates\ -d
C:\Chimera\Templates\tomograms.pkl
```

This command will train an SVM on the templates in C:\Chimera\Templates and the tomograms save in C:\Chimera\Templates\tomograms.pkl.
The result is saved in my_svm.pkl.

The format of the result is a pickle with the same structure as the return value of `svm_train` as mentioned in the python API

4. Evaluating tomograms.
   The usage documentation for the evaluation shell command is as follows:

```
usage: Main.py eval -d datapath -o out_path svm

positional arguments:
  svm                   Path to the pickle of the svm to use. (As returned by
                        the train subcommand)

optional arguments:
  -d datapath, --datapath datapath
                        Path to the tomograms to be evaluated.
  -o out_path, --outpath out_path
                        Path to which the results will be saved.
```

   A possible run of this command is as follows:

```
$> python Main.py eval my_svm.pkl -d C:\Chimera\Templates\tomograms.pkl -o
my_result.pkl
```

   This command evaluates the tomograms in C:\Chimera\Templates\tomograms.pkl and saves the results in my_result.pkl. Note that the evaluated tomogram needs to be a pickle of a list of tomograms as generated by the generation command.

   The format of the saved result is a pickle of the following tuple:

```
(svm_and_templates[1][0], tomograms, svm_and_templates[1][1], eval_result)
```

   The items of the pickle are as follows:
   The first item is the information about the tilts as saved by `svm_train`.
   `tomograms` is the list of tomograms as supplied to the command.
   The third item is the templates as saved by `svm_train`.
   `eval_result` is the evaluation result of all the tomograms as returned from `svm_eval`.

5. Viewing the results.
   The usage documentation for the view results shell command is as follows:

```
usage: Main.py view_results result

positional arguments:
  result      Path to the pickle of the result to show. (As returned by the
              eval subcommand)
```

   A possible run of this command is as follows:

```
$> python Main.py view_results my_result.pkl
```

This command only opens result from my_result.pkl and prints to the console a summary of the evaluation result of the first tomogram only.

Example

See examples/shell_flow_example

FlowTest

FlowTest is an all-in-one script that does all the steps automatically. Its behavior can be modified using a configuration file. The configuration file that will be used by the script can be set in two ways.
When running the script, if an additional argument is supplied it will be used as the path to the configuration file. Otherwise the variable `config_path` at the top of the script will be used as the path to the configuration.

Structure of the configuration for FlowTest

The configuration file is constructed out of three sections, default, train and evaluate, as can be seen below.
Example file:

```
[DEFAULT]
dim = 3
seed = 0
train = True
add_noise = True
svm_path = ..\TrainedSVM\5_Noise_3D_PDB_555.pkl
template_file_path = .\Templates\
angle_res = 60
generate_templates = True
templates_type = PDBS_3D

[train]
criteria = [5, 5, 5]
number_of_tomograms = 1

[evaluate]
criteria = [5, 5, 5]
number_of_tomograms = 1
metrics_input_file = ..\SVM_Metrics_Results\1_Noise_3D_PDB_555.pkl
metrics_output_file = ..\SVM_Metrics_Results\1_Noise_3D_PDB_555.pkl
```

The values in the file are as follows:
Values in the default section
dim – Integer value in {2, 3}. The dimension that will be used dimension by the FlowTest throughout.
seed – Integer value. The seed to be used by FlowTest. Can be deleted if not required.
train – Boolean value. This variable indicates whether the SVM should be trained or loaded (if it was previously trained for instance).
add_noise – Boolean value. This variable indicates whether generated tomograms should use noise in their generation.
svm_path – String value. This is the path (Absolute or relative to the directory from which the FlowTest script is being run, not necessarily the directory within which the code resides) to the SVM. If train is set to True, the new SVM will be saved to this path. Otherwise the path will be loaded as an SVM.

template_file_path – This is the path to the directory of the templates to be used. Only relevant if the dimension is set to 3.
angle_res – The angular resolution to be used in the generation of the templates (The Boolean value indicating whether to generate templates at all is in the train section)
generate_templates – Boolean value indicating whether new templates should be generated. If set to True while dimension is 3 then the templates that are generated will be saved into template_file_path.

Values in the train section. This entire section is only relevant if Train=True in the default section.
criteria – Python style list. This value will be used as is in the creation of the tomograms just like in the python API. Each number in the list of the criteria describes the number of templates that will be used of the template with the corresponding index.
num_of_tomograms – The number of tomograms that will be created for the training.

Values in the evaluation section
criteria – Python style list. Like in the train section, only here the criteria will be used for the generation of the tomograms which will be used in the evaluation. Each number in the list of the criteria describes the number of templates that will be used of the template with the corresponding index.
num_of_tomograms – The number of tomograms that will be created for evaluation.
metrics_input_file – If set, this path will be loaded as a previously created Metric object that will collect the evaluation results.
metrics_output_file – Path to which the result of the evaluation will be saved.

Note that the values and their names, as well as the names of the sections are case sensitive and should match the way they are seen in the example file above.

The Configuration File

The configuration file, by default config.txt, is used by the code to define its behavior. The configuration file can be changed manually in code using

```
from Configuration import CONFIG
CONFIG.load_config('path_to_my_config.txt')
```

The structure of the configuration file is as follows:

```
[config]

CHIMERA_PATH = C:\Program Files\Chimera 1.11.2\bin\chimera
CHIMERA_UTILS_PATH = C:\Software\src\ChimeraUtils\

DIM = 3
TEMPLATE_DIMENSION = 25
TOMOGRAM_DIMENSION = 120

# Noise parameters
NOISE_GAUSS_PARAM = 0.2
NOISE_LINEAR_PARAM = 0.4
NOISE_GAUSSIAN_SIZE = 5
NOISE_GAUSSIAN_STDEV = 3

# For correlation search
NEIGHBORHOOD_HALF_SIDE = 3

# Candidate selector peak detection constants
CORRELATION_THRESHOLD = 10
GAUSSIAN_SIZE = 10
GAUSSIAN_STDEV = 3

# For results metrics - if relative angle < TILT_THRESHOLD then correct tilt
TILT_THRESHOLD = 16

# For labeler
DISTANCE_THRESHOLD = 5
```

The values in the configuration file are as follows:

CHIMERA_PATH – This is the path to the chimera distribution. It should be set to full path to the chimera executable.

CHIMERA_UTILS_PATH – This is the path to the Chimera utility directory within the software. Replace the word “Software” with the full path to the project.

DIM – This is a variable for the dimension to be used. Note that only the shell API uses this value as the python API takes the dimension as parameter in the functions and FlowTest uses the dimension value specified in its own configuration file.

TEMPLATE_DIMENSION – This variable defines the size of the templates. Specifically, it is the side length of the templates such that, in the two dimensional case, the area of the templates will be the square of this variables value. This variable is currently not used for three dimensional creation of templates so three dimensional templates have a set size of 35 units to their side length.

TOMOGRAM_DIMENSION – This variable defines the size of the tomograms generated by the various generation functions. As with the template dimension, this variable is the side length of the tomograms. This variable, however, does affect both the two dimensional case as well as the three dimensional case.

NOISE_GAUSS_PARAM – Floating point value. This variable defines the strength, or amplitude, of the Gaussian part of the noise that is added to the generated tomograms when the noise is used.
NOISE_LINEAR_PARAM – Floating point value. This variable defines the coefficient of the linear noise that is added to the generated tomograms when the noise is used.
NOISE_GAUSSIAN_SIZE – Integer value. This variable defines the size of the Gaussian kernel that is used to blur the noise when it is used.
NOISE_GAUSSIAN_STDEV – Integer value. This variable defines the standard deviation of the Gaussian kernel that is used to blur the noise when it is used.
NEIGHBORHOOD_HALF_SIDE – Integer value. This variable defines the size of the space, or neighborhood, around a candidate within which the best match for a position and tilt will be searched. Specifically, the variable defines half of the side of a hypercube of the relevant dimension.
CORRELATION_THRESHOLD – Floating point value. Used as threshold for the correlation of the templates with the tomogram when selecting candidates in the CandidateSelector object.
GAUSSIAN_SIZE – Integer value. This variable defines the size of the Gaussian kernel that will be used for the selection of candidates in the CandidateSelector object.
GAUSSIAN_STDEV – Integer value. This variable defines the standard deviation of the Gaussian kernel in the CandidateSelector object.
TILT_THRESHOLD – Integer value. This variable defines the threshold, in degrees in the two dimensional case or solid angle in the three dimensional case, for the difference in direction between two different tilts of candidates.
DISTANCE_THRESHOLD – Floating point value. This variable is used in the PositionLabeler to distinguish between detections of matches that are false to those close to the ground truth. If the candidate is closer to the ground truth than the threshold dictates, the candidate is labeled as a true result.
This variable is also used in MetricTester as a minimum distance for finding the best of a candidate to a composition.