SENSAAS-Flex

Alignment of molecules using colored shape and conformational flexibility

References:

Biyuzan H., Masrour M.-A., Grandmougin L., Payan F. and Douguet D., SENSAAS-Flex: a joint optimization approach for aligning 3D shapes and exploring the molecular conformation space, Bioinformatics, 2024. Doi: 10.1093/bioinformatics/btae105

Douguet D. and Payan F., SenSaaS: Shape-based Alignment by Registration of Colored Point-based Surfaces, *Molecular Informatics*, **2020**, 8, 2000081. Doi: 10.1002/minf.202000081

Introduction:

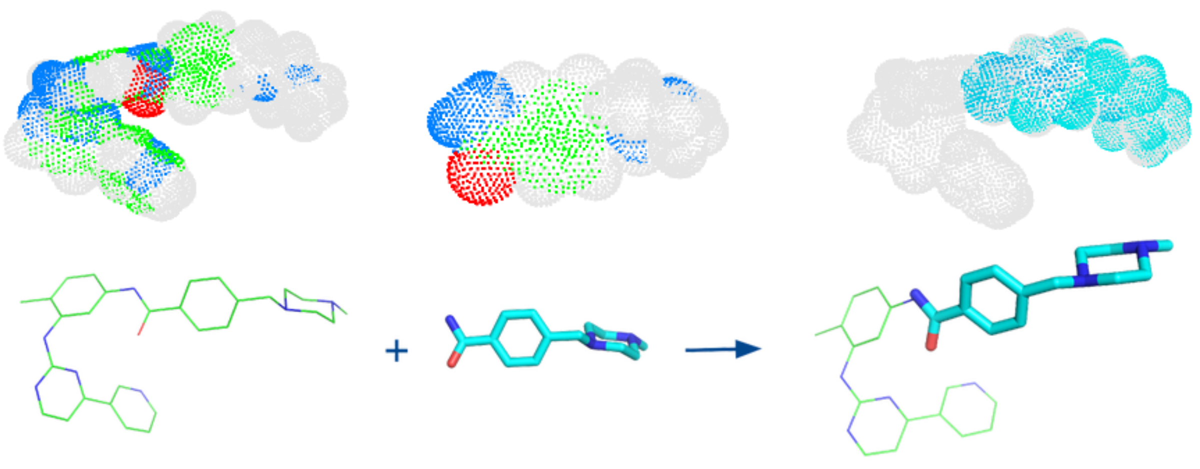
3D shape is an important molecular feature in biological activity. **SENSAAS** (SENsitive Surface As A Shape) compares molecules by aligning point-based representation of the van der Waals surface and provides superimposition of 3D graphs and similarity scores. Further, SENSAAS-Flex perform the flexible alignment of two molecular shapes by optimizing the 3D conformation of the aligned molecule.

The documentation of the first version of SENSAAS can be found on GitHub at https://github.com/SENSAAS/sensaas-py/blob/main/docs/index.rst. This version allows rigid alignment only.

The documentation of SENSAAS-Flex is provided below.

SENSAAS is a shape-based alignment software which allows to superimpose molecules

What is SENSAAS?



SENSAAS is the result of a collaboration between researchers of two labs of UniCA (University Côte d'Azur): I3S and IPMC. Based on the publication SenSaaS: Shape-based Alignment by Registration of Colored Point-based Surfaces, **SENSAAS** is a shape-based alignment software which allows to superimpose molecules in 3D space.

Tutorial: Several videos on YouTube provide tutorials for installing and executing SENSAAS

1. Requirements and installation https://www.youtube.com/watch?v=2mYZlW4QbvQ
2. Using sensaas, a basic example https://www.youtube.com/watch?v=IvLDvVvfMTA
3. Using meta-sensaas for virtual screening https://www.youtube.com/watch?v=Z3qLQXEbW8o
4. Using meta-sensaas for clustering https://www.youtube.com/watch?v=X5caj1us6rY

Our algorithm runs with Python and requires the open-source library Open3D.

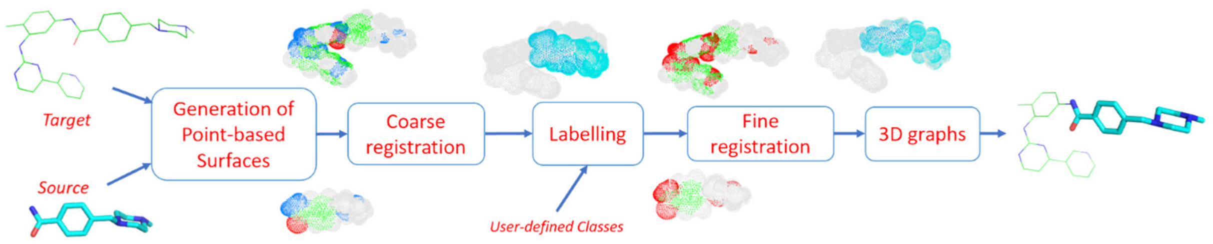
How does SENSAAS work?

3D point clouds or 3D meshes are data structures used in many fields well known by large audience (robotics, 3D reconstruction, games, autonomous navigation...) to model surfaces or volumes. Such 3D representations can also be relevant in chemistry to describe molecules although they were not the most used so far.

SENSAAS (SENsitive Surface As A Shape) is a shape-based alignment software using 3D point-based representation of the van der Waals surface. SENSAAS is an original tool that combines recent methods dedicated to 3D registration, initially developed for the fusion of 3D point clouds, collected by devices such as depth cameras or LiDAR scanners.

Considering two molecules named Source and Target as inputs, SENSAAS gives a transformation matrix as output, leading to the "best" 3D alignment of Source on Target. SENSAAS follows four major steps:

- generation of a point cloud from the molecular surface of each input molecule
- coarse alignment of the two point clouds thanks to a geometry-aware global registration
- labelling of each point of the two clouds according to user-defined classes
- refinement of this alignment by applying a color and geometry-aware local registration. At this step, a color is assigned to each point, in function of its label



1. Generation of input point clouds For each input file (Source and Target), a point cloud of the van der Waals surface is obtained. Each point is described by its 3D coordinates, and a color (RGB) according to the nature of the underlying atom.

2. Coarse alignment by global registration the Source point cloud is globally superimposed on the Target, by finding an initial matching in terms of geometry only. The matching is done by using local 3D descriptors computed on a limited number of points (also called downsampled point clouds). We use the descriptors FPFH (presented in Fast Point Feature Histograms (FPFH) for 3D Registration), and the matching is done with RANSAC (Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography).

3. Labelling of the points of each point cloud Each point is colored according to its belonging to a user-defined class. In the current version, the classes depend on the pharmacophore features, but they could depend on any physico-chemical property mapped onto the surface, or any configuration given by the user.

4. Refinement of the alignment At this step, the registration takes into account the geometry of the point clouds, but also the color of the points assigned by labelling. The coarse alignment is improved by finding the best matching between the two colored point clouds. The method used here is the method of Colored Point Cloud Registration Revisited. This step results into a final transformation matrix (rotation + translation), that is applied to the Source molecule to get the final alignment.

Installing

SENSAAS relies on the open-source library Open3D. The current release of SENSAAS uses **Open3D version 0.12.0** along with **Python3.7**.

Visit the following URL for using Python packages distributed:

- via PyPI: http://www.open3d.org/docs/release/getting_started.html
- or conda: https://anaconda.org/open3d-admin/open3d/files. For example, for windows-64, you can download *win-64/open3d-0.12.0-py37_0.tar.bz2*

Virtual environment for python with conda (for Windows for example)

Install conda or Miniconda.

Launch Anaconda Prompt, then complete the installation:

```
conda update conda
conda create -n sensaas
conda activate sensaas
(sensaas) > conda install python=3.7 numpy
```

Once Open3D downloaded:

```
(sensaas) > conda install open3d-0.12.0-py37_0.tar.bz2
```

Some scripts written in Perl are used in SENSAAS-Flex.

```
(sensaas) > conda install perl
```

As well, the open-Source Cheminformatics Software RDKit must be installed.

```
(sensaas) > conda install -c conda-forge rdkit
```

(Optional) Install additional packages for visualization with PyMOL (more information at https://pymolwiki.org):

```
(sensaas) > conda install -c schrodinger pymol
```

If an error regarding PyQT occurs then:

```
(sensaas) > conda install -c anaconda pyqt
```

Retrieve and unzip SENSAAS-Flex repository in your desired folder. The directory containing executables is called sensaas-flex-main. See below for running the programs **sensaas.py, meta-sensaas.py**, **sensaasflex.py and meta-sensaasflex.py**.

Linux

Scripts written in Perl are used in SENSAAS-Flex, thus, check for a Perl package on your system.

Install (more information at http://www.open3d.org/docs/release/getting_started.html):

```
1. Python 3.7 and numpy
2. Open3D version 0.12.0
```

The open-Source Cheminformatics Software RDKit must be installed. More information on RDKit can be found at https://www.rdkit.org/docs/Install.html

```
3. RDKit
```

(Optional) Install additional packages for visualization with PyMOL (more information at https://pymolwiki.org):

```
4. PyMOL
```

Retrieve and unzip SENSAAS-Flex repository in your desired folder. The directory containing executables is called sensaas-flex-main. See below for running the programs **sensaas.py, meta-sensaas.py**, **sensaasflex.py and meta-sensaasflex.py**.

MacOS

Not tested

List of I/O Formats

In our implementation, input molecules are **3D structures with explicit hydrogen atoms**. Molecules are represented either by their 3D graphs or by their resulting 3D point clouds.

SENSAAS's scripts can read several input file formats:

Input type	**File format**	
sdf	SDF format file	**3D graph**
pdb	PDB format file	(**3D graph**) reads ATOM and HETATM coordinates
dot	PDB format file	(**Point cloud**) reads HETATM lines that contain coordinates of dots and the atom type for defining the label
xyzrgb	xyzrgb format file	(**Point cloud**) ascii file used in 3D data processing such as Open3D; contains coordinates of dots and color
pcd	PCD format file	(**Point cloud**) used in 3D data processing such as Open3D

The output file format depends on the input file format:

- if the Source input file is **sdf** then **Source_tran.sdf** is the transformed sdf source file
- if the Source input file is **pdb** then **Source_tran.pdb** is the transformed pdb source file
- if the Source input file is **dot** then **Source-dots_tran.pdb** is the transformed dot file in pdb format
- if the Source input file is **xyzrgb** then **Source_tran.xyzrgb** is the transformed xyzrgb file
- if the Source input file is **pcd** then **Source_tran.pcd** is the transformed pcd file

Colors

In our implementation, labels aim to recapitulate typical pharmacophore features such as aromatic (colored in green), lipophilic (colored in white/grey) and polar groups (colored in red):

- **class 1** (or label 1) includes non polar hydrogen (H) and halogen atoms excepting fluorines (Cl, Br and I). Hydrogen and halogen atoms are molecule endings. They are the most frequent atoms that contribute to the surface geometry and coloration, and thus, highlight the apolar surface area. Points belonging to this class are colored in white/grey.
- **class 2** (or label 2) includes polar atoms able to be involved in hydrogen bonds such as N, O, S, H (if linked to N or O) and F. Points belonging to this class are colored in red.

- **class 3** (or label 3) includes “skeleton elements” such as C, P and B. Points belonging to this class are colored in green.
- **class 4** (or label 4) includes all elements not listed in the first three classes. This class is empty for most small organic molecules in medicinal chemistry. Points belonging to this class are colored in blue.

Fitness scores

The alignments provided by SENSAAS are evaluated by fitness scores calculated from point clouds. A fitness score indicates how many points are paired. Points are considered paired if their distance is lower than a given threshold. In our implementation, we set the threshold value to 0.3 because it is the average distance between two adjacent points in our original point clouds.

Each score is similar to a Tversky coefficient tuned to evaluate the embedding of a point cloud in another one. Therefore, the score of the Source and the score of the Target may differ. The smallest point cloud of the two will always obtain the highest fitness score as more points are paired, proportionally.

There are three different fitness scores, but we only use 2 of them, gfit and hfit, to finally calculate gfit+hfit.

- **gfit** estimates the geometric matching of point-based surfaces. It is the ratio between the number of points of the transformed Source that match points of the Target, and its total number of points - **it ranges between 0 and 1**
- **hfit** estimates the matching of colored points representing pharmacophore features. It is the sum of the fitness for each class except the first class, to specifically evaluate the matching of polar and aromatic points (classes 2, 3 and 4) - **it ranges between 0 and 1**
- cfit is the sum of the fitness for each class, to specifically evaluate the matching of the colored points of the 4 classes - it ranges between 0 and 1

The hybrid score called **gfit+hfit** is the sum = gfit + hfit scores **- gfit+hfit ranges between 0 and 2**

- A gfit+hfit score close to 2.0 means a perfect superimposition.
- A gfit+hfit score close to 0.0 means that there are no similarities between molecular shapes.

Tutorials

This tutorial presents several basic usages of SENSAAS with sdf molecular files. In the folder examples, you will find some molecular files as test cases to work with.

P04035-7.sdf is the experimental conformer of the ligand 4HI (statin inhibitor) observed in the protein-ligand complex PDB 3CCW (3-hydroxy-3-methylglutaryl coenzyme A reductase (HMGR); hmg-coa reductase). This example was extracted from the AstraZeneca data set.

P04035-7-confs.sdf contains 10 conformers generated with rdkit.

P04035-7-confs1.sdf is a conformer generated with rdkit.

phenyl.sdf contains a fragment that will be tested in the clustering mode.

I - Run a rigid alignment with sensaas.py

This script allows to align one Source molecule on one Target molecule:

```
sensaas.py <target-type> <target-file-name> <source-type> <source-file-name>
<log-file-name> <mode>
```

<target-type>

type of the Target file (sdf/pdb/dot/xyzrgb/pcd)

<target-file-name>

name of the Target file

<source-type>

type of the Source file (sdf/pdb/dot/xyzrgb/pcd)

<source-file-name>

name of the Source file

<log-file-name>

name of the log file that details the alignment with **scores of Source**.

<mode>

- **optim** executes the alignment and generates a transformation matrix
- **eval** evaluates the superimposition "in place" (without aligning)

a) Example with the ‘optim’ mode

The following example works with two molecules from the directory examples/

```
sensaas.py sdf P04035-7.sdf sdf P04035-7-confs1.sdf slog.txt optim
```

Note:
Don't worry if you get the following warning from Open3D: "*Open3D WARNING KDTreeFlann::SetRawData Failed due to no data.*". It is observed with conda on windows.

Here, the source file P04035-7-confs1.sdf is aligned (**moved**) on the target file P04035-7.sdf (experimental conformation) (**that does not move**).

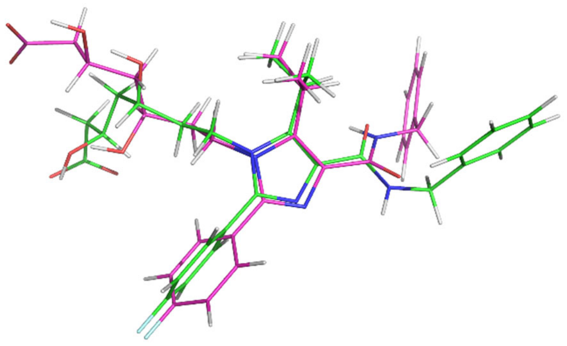
- The output file **Source_tran.sdf** contains the aligned (transformed) coordinates of the Source.
- The output file **tran.txt** contains the transformation matrix applied to the input Source file.
- The **slog.txt** file details results with final scores of the aligned Source molecule on the last line. In the current example, the last line may look like:

  “`gfit= 0.356 cfit= 0.302 hfit= 0.252 gfit+hfit= 0.608`”

It indicates that the best alignment of the Source file (P04035-7-confs1.sdf) has a fitness score gfit+hfit = 0.61.

Visualization To visualize the result, you can use any molecular viewer. For instance, you can use PyMOL if installed (see optional packages) to load the Target and the aligned Source:

```
pymol P04035-7.sdf Source_tran.sdf
```



The Source_tran.sdf (in magenta) is aligned on the experimental conformation P04035-7.sdf (in green). The RMSD value between conformers equals 3.36 Å (see RMSD calculation below).

RMSD calculation If two molecules are exactly the same then they possess the same 3D graph. In such case, the root-mean-square distance (RMSD) of corresponding atom pairs in 3D graphs can be calculated using two RDKit based tools (both are present in the folder utils).

The first one is SymmFit (author: Paolo Tosco; the code can be found at https://www.mail-archive.com/rdkit-discuss@lists.sourceforge.net/msg04915.html) which allows the minimization of RMSD value but only when the atoms in the two structure files are arranged in the same order.

Syntax (‘in place’ RMSD calculation):

```
rdkit-symmFitRMSD.py -r mol1.sdf mol2.sdf
```

Syntax (minimizes RMSD and writes transformation matrix called rdkit-tran.txt):

```
rdkit-symmFitRMSD.py -s mol1.sdf mol2.sdf
```

The second one is CalcLigRMSD (author: Carmen Esposito; the code can be found at https://github.com/cespos/rdkit/tree/add-CalcLigRMSD-for-prealigned-compounds/Contrib/CalcLigRMSD) which allows the calculation of RMSD value of ‘in place’ structures (without minimization) whatever the arrangement of atoms in the two structure files.

In the present case study, we use that one because our atoms are not in the same order in the sdf files. We obtain:

```
rdkit-CalcLigRMSD.py P04035-7.sdf Source_tran.sdf
RMSD= 3.36
```

b) Example with the ‘eval’ mode:

Given two molecules, molecule1.sdf and molecule2.sdf, the eval mode evaluates the superimposition "in place" (without aligning)

```
sensaas.py sdf molecule1.sdf sdf molecule2.sdf slog.txt eval
```

Here, the resulting slog.txt contains final scores of molecule2.sdf on the last line.

```
sensaas.py sdf molecule2.sdf sdf molecule1.sdf slog.txt eval
```

Here, the resulting slog.txt contains final scores of molecule1.sdf on the last line.

In our present case study, we can recalculate fitness scores of the aligned Source_tran.sdf (see slog.txt):

```
sensaas.py sdf P04035-7.sdf sdf Source_tran.sdf slog.txt eval
```

or calculate the score of the Target P04035-7.sdf (see slog.txt):

```
sensaas.py sdf Source_tran.sdf sdf P04035-7.sdf slog.txt eval
```

II - Run rigid alignment(s) with meta-sensaas.py

This "meta" script only works with SDF format files. It allows to align several Source and Target molecules.

The syntax is:

```
meta-sensaas.py molecules-target.sdf molecules-source.sdf -r nb -s score_type
-l x
```

<molecules-target.sdf>

name of the Target file

<**molecules-source.sdf**>

name of the Source file

<-s score_type>

Choose the score type {source target mean}:

-s source
here the score of the aligned source will be used to rank solutions and fill matrix-sensaas.txt. This is the default setting if the option -s is not indicated.

-s target
here the score of the target will be used to rank solutions and fill matrix-sensaas.txt.

-s mean
here the mean of the score of the target and of the aligned source will be used to rank solutions and to fill matrix-sensaas.txt. This option is interesting to favor source molecules that have the same size of the Target.

<-l x>

This option sets the database/virtual screening mode. It allows to repeat (loop) x times the alignment using SENSAAS. x is an integer. As SENSAAS is based on stochastic algorithms, one may want to repeat, for example, 2 times a calculation. Results show that it improves alignments when molecules have few similarities. Only the best result is kept.

<-r nb>

Used to execute the clustering mode. nb is an integer. It indicates the number of repeats of SENSAAS before clustering the alignments.

a) Database/Virtual Screening mode

This script is suited to run virtual screenings of sdf files containing several molecules (database mode). For example, if you want to screen a sdf file containing several conformers for Target and/or Source. Solutions can then be ranked in descending order of score and a similarity matrix is provided. The syntax is:

```
meta-sensaas.py molecules-target.sdf molecules-source.sdf
```

Example

```
meta-sensaas.py P04035-7.sdf P04035-7-confs.sdf
```

Note:
Don't worry if you get the following warning from Open3D: "*Open3D WARNING KDTreeFlann::SetRawData Failed due to no data.*". It is observed with conda on windows.

Outputs are:

- the file **bestsensaas.sdf** that contains the best ranked aligned Source
- the file **catsensaas.sdf** that contains all aligned Sources in the order in which the input file is read
- the file **matrix-sensaas.txt** that contains gfit+hfit scores (rows=Targets and columns=Sources)

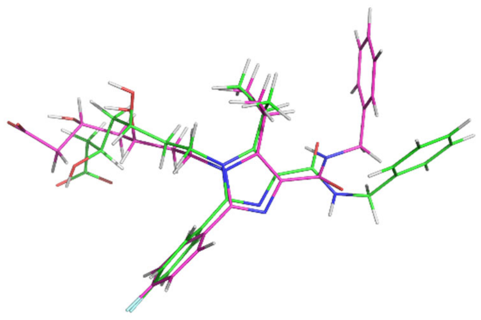
The last line of the output at your prompt may look like:

```
“gfithfit= 0.659 gfit= 0.389 hfit= 0.270 (target 1 ; source 5 ; outputs:
bestsensaas.sdf, catsensaas.sdf and matrix-sensaas.txt)
```

It indicates that the best Source solution is the molecule number 5 of the file P04035-7-confs.sdf. This best aligned conformer has a gfithfit score = 0.66.

Visualization You can use any molecular viewer. For instance, you can use PyMOL if installed.

```
pymol P04035-7.sdf bestsensaas.sdf catsensaas.sdf
```



The file bestensaas.sdf (in magenta) is aligned on the experimental conformation P04035-7.sdf (in green).

RMSD calculation

```
rdkit-CalcLigRMSD.py P04035-7.sdf bestsensaas.sdf
RMSD= 2.42
```

Post-processing To ease the analysis of the results, the script **ordered-catsensaas.py** found in the directory utils can be used to generate files in descending order of score.

```
ordered-catsensaas.py matrix-sensaas.txt catsensaas.sdf
```

or if you want to only retrieve solutions having a gfit+hfit score above a defined cutoff; for example, to select only solutions having a gfit+hfit score >= 0.6:

```
ordered-catsensaas.py matrix-sensaas.txt catsensaas.sdf 0.6
```

- the file **ordered-catsensaas.sdf** contains all aligned Sources in descending order of score
- the file **ordered-scores.txt** contains the original number of Source with gfit+hfit scores in descending order

Visualization You can use any molecular viewer. For instance, you can use PyMOL if installed.

```
pymol P04035-7.sdf ordered-catsensaas.sdf
```

b) Clustering mode in order to find alternative alignments

This option allows to repeat alignments in order to find alternative superimpositions when they exist as for example when aligning a fragment on a large molecule. Here, clusters are created based on their distance. It works with one Target and one Source only (the first molecule of each input sdf file is read).

Example

```
meta-sensaas.py P04035-7.sdf phenyl.sdf -r 10
```

Note:
Don't worry if you get the following warning from Open3D: "*Open3D WARNING KDTreeFlann::SetRawData Failed due to no data.*". It is observed with conda on windows.

Here 10 alignments of the Source will be generated and clustered.

Outputs are:

- the file **sensaas-1.sdf** with the best ranked alignment - it contains 2 molecules: first is Target and second the aligned Source
- the file **sensaas-2.sdf** (if exists) with the second best ranked alignment - it contains 2 molecules: first is Target and second the aligned Source
- ...
- file **cat-repeats.sdf** that contains all aligned Sources

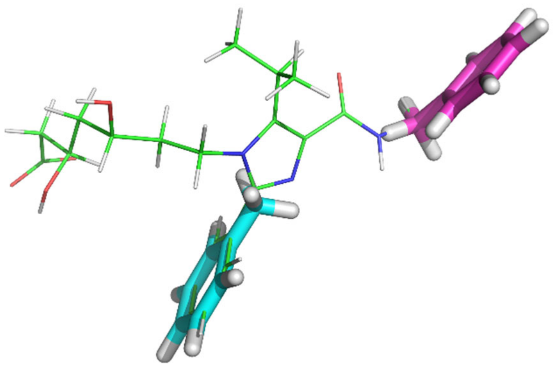
The last line of the output at your prompt may look like:

```
Rank ; Hybrid score gfit + hfit ; Shape gfit ; Color hfit ; percentage of
solutions ; Alignment in SDF file format
1 ; 1.885 ; 0.907 ; 0.977 ; 90.00 ; sensaas-1.sdf
2 ; 1.522 ; 0.692 ; 0.831 ; 10.00 ; sensaas-2.sdf
cat-repeats.sdf contains the 10 sdf solutions
```

It indicates that 2 clusters were created. File sensaas-1.sdf is the best solution of cluster 1 and sensaas-2.sdf is the best solution of cluster 2. Both solutions are perfectly aligned on substructures of P04035-7.sdf.

Visualization You can use any molecular viewer. For instance, you can use PyMOL if installed.

```
pymol P04035-7.sdf sensaas-1.sdf sensaas-2.sdf cat-repeats.sdf
```



The file sensaas-1.sdf (in magenta) and sensaas-2.sdf (in cyan) are aligned on the experimental conformation P04035-7.sdf (in green).

III - Run a flexible alignment with sensaasflex.py

This "meta" script only works with SDF format files. This script allows to run flexible alignment of two shapes by optimizing the conformer of the Source.

The syntax is:

```
sensaasflex.py <target sdf file (read first structure only)> <source sdf file
(to move; read first structure only)>
```

Example

```
sensaasflex.py P04035-7.sdf P04035-7-confs1.sdf
```

Here, the source file P04035-7-confs1.sdf is aligned (**moved**) on the target file P04035-7.sdf (experimental conformation) (**that does not move**).

- The output file **Source_tran.sdf** contains the aligned (transformed) coordinates of the Source.
- The output file **tran.txt** contains the transformation matrix applied to the input Source file.

- The **slogflex** file details results with final scores of the aligned Source molecule on the last line. The content of the slogflex file may look like (of note, the result may vary from one run to another):

```
“Initial gfit= 0.338 cfit= 0.289 hfit= 0.252 gfit+hfit= 0.59
Round 1 gfit= 0.38 cfit= 0.307 hfit= 0.196 gfit+hfit= 0.576
Round 2 gfit= 0.695 cfit= 0.649 hfit= 0.557 gfit+hfit= 1.252
Best solution (Source_tran.sdf):
gfit= 0.695 cfit= 0.649 hfit= 0.557 gfit+hfit= 1.252”
```

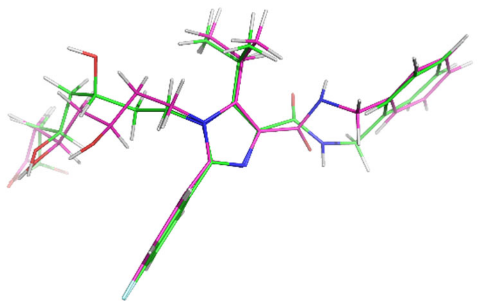
Here, it indicates that the initial rigid alignment gives a solution with a gfit+hfit score = 0.59 and that the alignment obtained after the flexible optimization has a gfit+hfit score = 1.25 which is a much better fitness score.

RMSD calculation (the two molecules are the same; they possess the same 3D graph)

```
rdkit-CalcLigRMSD.py P04035-7.sdf Source_tran.sdf
RMSD= 1.04
```

Visualization You can use any molecular viewer. For instance, you can use PyMOL if installed.

```
pymol P04035-7.sdf Source_tran.sdf
```



The Source_tran.sdf (in magenta) is aligned on the experimental conformation P04035-7.sdf (in green). The RMSD value between conformers equals 1.04Å.

Here we show that SENSAAS-Flex was able to explore the conformational space to find a very close solution to the experimental one. It improves the alignment over the rigid one. Indeed, the first rigid alignment using **sensaas.py** shown above had a gfit+hfit score of 0.608 and a RMSD value of 3.36 Å.

IV - Run flexible alignment(s) with meta-sensaasflex.py

As for meta-sensaas.py, this "meta" script only works with SDF format files. It allows to align several source and target molecules.

The syntax is:

```
meta-sensaasflex.py  molecules-target.sdf  molecules-source.sdf  -r  nb  -s
score_type -l x
```

Please read the above description of **meta-sensaas.py** for more information about options and outputs (paragraph "**II - Run rigid alignment(s) with meta-sensaas.py**").

Miscellaneous Tools

If you want that **sensaas.py** outputs Target and Source files in pcd and xyzrgb format, set the variable 'verbose' to 1 in the Python script **sensaas.py**. Those format files can be read and visualized using Open3D.

For example:

```
utils/visualize.py examples/VALSARTAN.xyzrgb
```

or:

```
utils/visualize.py examples/VALSARTAN.pcd
```

You can also convert a xyzrgb file into a pdb file for visualization with PyMOL

```
utils/xyzrgb2dotspdb.py examples/VALSARTAN.xyzrgb
```

It will generate the file 'dots.pdb' that can be read with the molecular viewer PyMOL:

```
pymol dots.pdb
```

Colors are class colors as defined above. In our implementation, labels aim to recapitulate typical pharmacophore features such as aromatic (colored in green), lipophilic (colored in white/grey) and polar groups (colored in red).

Information on differences between 0.7.0 and 0.12.0 versions of Open3D

Those differences are tagged in python scripts **sensaas.py sensaasCICP.py GCICP.py** and **CICP.py**. Differences are:

evaluate_registration

```
#open0.7
score=o3d.registration.evaluate_registration(source_pcd,target_pcd,threshold,tran)

#open0.12 or later add ".pipelines"
score=o3d.pipelines.registration.evaluate_registration(source_pcd,target_pcd,threshold,tran)
```

voxel_down_sample

```
#open0.7
pcd_down = o3d.geometry.voxel_down_sample(pcd, voxel_size)

#open0.10 or later
pcd_down = pcd.voxel_down_sample(voxel_size)
```

estimate_normals

```
#open0.7
o3d.geometry.estimate_normals(pcd_down, o3d.geometry.KDTreeSearchParamHybrid(radius = radius_normal, max_nn = 30))

#open0.10 or later
pcd_down.estimate_normals(o3d.geometry.geometry.KDTreeSearchParamHybrid(radius = radius_normal, max_nn = 30))
```

And

```
#open0.7
o3d.geometry.estimate_normals(source_down,
o3d.geometry.KDTreeSearchParamHybrid(radius = voxel_size * 2, max_nn = 30))
o3d.geometry.estimate_normals(target_down,
o3d.geometry.KDTreeSearchParamHybrid(radius = voxel_size * 2, max_nn = 30))

#open0.10 or later
source_down.estimate_normals(o3d.geometry.KDTreeSearchParamHybrid(radius = voxel_size * 2, max_nn = 30))
target_down.estimate_normals(o3d.geometry.KDTreeSearchParamHybrid(radius = voxel_size * 2, max_nn = 30))
```

compute_fpfh_feature

```
#open0.7
pcd_fpfh = o3d.registration.compute_fpfh_feature(pcd_down,o3d.geometry.KDTreeSearchParamHybrid(radius = radius_feature, max_nn = 100))
```

```
#open0.10 or later
pcd_fpfh                                                                                         =
o3d.pipelines.registration.compute_fpfh_feature(pcd_down,o3d.geometry.KDTreeSearchPara
mHybrid(radius = radius_feature, max_nn = 100))
```

registration_ransac_based_on_feature_matching

```
#open0.7
result = o3d.registration.registration_ransac_based_on_feature_matching(
        source_down, target_down, source_fpfh, target_fpfh,
        distance_threshold,
        o3d.registration.TransformationEstimationPointToPoint(False), 4,
        [o3d.registration.CorrespondenceCheckerBasedOnEdgeLength(0.9),
        o3d.registration.CorrespondenceCheckerBasedOnDistance(distance_threshold)],
        o3d.registration.RANSACConvergenceCriteria(400000, 1000))

#open0.10 or later
result = o3d.pipelines.registration.registration_ransac_based_on_feature_matching(
        source_down, target_down, source_fpfh, target_fpfh, True,
        distance_threshold,
        o3d.pipelines.registration.TransformationEstimationPointToPoint(False), 4,
        [o3d.pipelines.registration.CorrespondenceCheckerBasedOnEdgeLength(0.9),

o3d.pipelines.registration.CorrespondenceCheckerBasedOnDistance(distance_threshold)],
        o3d.pipelines.registration.RANSACConvergenceCriteria(400000, 1000))
```

registration_colored_icp

```
#open0.7
result    =    o3d.registration.registration_colored_icp(source_down,    target_down,radius,
globaltran,
o3d.registration.ICPConvergenceCriteria(relative_fitness = 1e-6,
        relative_rmse = 1e-6, max_iteration = max_iter),lambda_geometric=0.8)


#open0.10 or later
result = o3d.pipelines.registration.registration_colored_icp(source_down, target_down,radius,
globaltran,
 o3d.pipelines.registration.ICPConvergenceCriteria(relative_fitness = 1e-6,
        relative_rmse = 1e-6, max_iteration = max_iter),lambda_geometric=0.8)
```

//