LOOS: A Tool for Making New Tools for Analyzing Molecular Simulations

Tod D. Romo and Alan Grossfield

University of Rochester Medical Center, Rochester, NY, USA

http://loos.sourceforge.net

LOOS Information & Download

J Comp Chem 35, 2305 (2014)

LOOS Paper

Abstract

We have developed LOOS (Lightweight Object Oriented Structure-analysis) as a tool for making new tools to analyze molecular simulations. LOOS is an object-oriented library designed to facilitate the rapid development of new methods for structural analysis. LOOS is written in C++ and is easily extensible, only requiring knowledge of 4 core classes. A Python interface is also available, further facilitating rapid development of analysis tools and broadening the LOOS community. LOOS supports reading the native file formats of most common simulation packages and can write NAMD formats (PDB and DCD) and Gromacs XTC. A dynamic atom selection language, based on C expression syntax, is included and is easily accessible via a single function call. LOOS includes over 140 pre-built tools for common structural analysis tasks, including analyzing simulation convergence, 3D histograms, and elastic network models. Python-based packages include the *OptimalMembraneGenerator* and tools for the Voronoi tesselation of membranes. In addition, tool templates for common analysis work-flows are included to facilitate the development of new tools. LOOS is available for download at **loos.sourceforge.net** or can be cloned from **github.com/GrossfieldLab/loos**, and is distributed under the GPLv3 license.

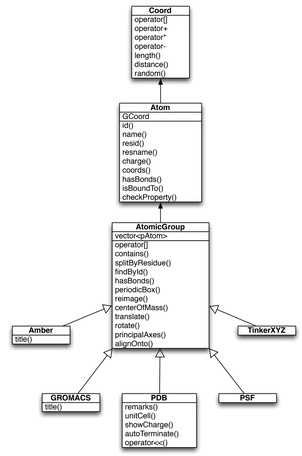
Design Goals

- **Lightweight**
  - Tool developers only need 4 core classes: `Coord`, `Atom`, `AtomicGroup`, `Trajectory`
  - Few external dependencies: Boost, scons, atlas/lapack SWIG & NetCDF
  - All available via package system on Linux
- **Extensible**
  - Polymorphic classes
  - Algorithm encapsulation
  - Design patterns for easy extension
- **Powerful**
  - Rich atom selection language
  - Parser built using standard Unix tools
  - Many useful member functions
  - Shared atoms via Boost shared pointers
    - Simplifies memory management
    - Copies are lightweight
  - Standard Template Library support
  - Support for basic periodicity
- **Easy to use**
  - Complex tools with minimal code
  - Python interface to core library
  - Exceptions translated into Python
  - Rapid development of new tools
  - Templates for writing new tools for common cases
  - Tools are self-documenting
- **Multiplatform Support**
  - All major Linux distributions
    - Tested on distros up to 4 years old
  - MacOS X
- **Multipackage Support**
  - CHARMM/NAMD
  - Amber (including NetCDF)
  - GROMACS/MARTINI
  - Tinker
  - Outputs PDB, DCD, and XTC
  - Easy to extend

Class Structures

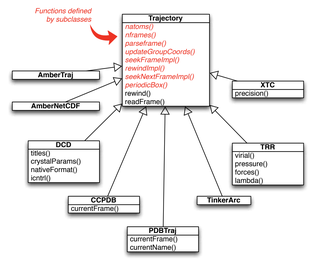
Defining The System (Model)

Not all functions listed

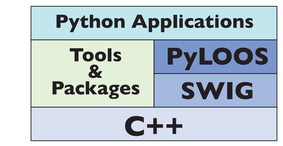


Trajectories

Not all functions listed



Application Layers



Example Code

Tracking Motion of Two Protein Segments

```
#!/usr/bin/env python
import sys
import math
import loos
import loos.pyloos
```

Read Command Line

```
system_file = sys.argv[1]
traj_file = sys.argv[2]
sel_string1 = sys.argv[3]
sel_string2 = sys.argv[4]
frames_to_skip = int(sys.argv[5])
```

Create System

```
system = loos.createSystem(system_file)
traj = loos.pyloos.Trajectory(traj_file, system,
                              skip = frame_to_skip)
```

Select "Domains"

```
sel1 = loos.selectAtoms(system, sel_string1)
sel2 = loos.selectAtoms(system, sel_string2)
```

Loop Over Trajectory

```
for frame in traj:
```

Compute Distance

```
    # Compute distance
    centroid1 = sel1.centroid()
    centroid2 = sel2.centroid()

    diff = centroid2 - centroid1
    distance = diff.length()
```

Compute Angle

```
    # Compute angle between principal axes
    vectors1 = sel1.principalAxes()
    axis1 = vectors1[0]

    vectors2 = sel2.principalAxes()
    axis2 = vectors2[0]
    angle = math.acos(axis1 * axis2) * 180/math.pi
```

Compute Torsion

```
    # Compute torsion between principal axes
    p1 = centroid1 + axis1
    p2 = centroid2 + axis2

    tors = loos.torsion(p1, centroid1, centroid2, p2)

    # Write output
    print traj.realIndex(), distance, angle, tors
```

Bundled Tools

Over 140 tools total, including 7 packages and 70 core tools

Sample of tools included in LOOS

Core Tools	
aligner	Optimally align trajectory
contact-time	Time-series of atom contacts
density-dist	Electron, mass, or charge density along the z-axis
merge-traj	Merge & subsample trajectories
order_parameters	Order parameters analogous to ^{2}H quadrupolar splitting for lipid chains.
rdf	Radial distribution function
rmsds	All-to-all RMSD
svd	Singular Value Decomposition of a trajectory (PCA)
xy_rdf	Radial distribution function in the x,y-plane
Convergence Package	
block_average	Block average of arbitrary time-series data
coscon	Cosine content of a trajectory
decorr_time	Decorrelation time of a trajectory
bcom, boot_bcom	Block Covariance Overlap Method for determining convergence & sampling
Gridded Density Package	
grid2xplor	Convert density grid to X-plor electron density map format for visualization
near_blobs	Find residues that are near a blob for a trajectory
water-hist	Density histogram for atoms in a trajectory
Elastic Network Package	
anm	Anistropic Network Model
enmovie	Visualize ENM motions by generating a trajectory for an ENM solution
vsa	Vibrational Subsystem Analysis
Hydrogen Bonds Package	
hbonds	Find occupancies of putative hydrogen bonds in a trajectory
hcontacts	Time-series of possible intra- and inter-molecular hydrogen bonds
hcorrelation	Time-correlation of putative hydrogen bonds
Optimal Membrane Generator Package	
OptimalMembraneGenerator.py	Build membrane de novo with multiple lipid types, protein, cholesterol, asymmetric bilayers, etc
Voronoi Package	
area_per_molecule.py	Distribution of area/molecule for a z-slice using 2D Voronoi
area_profile.py	Voronoi cross-sectional area for object through a membrane

Selection Language

- Based upon C/C++ expressions
- Built using lex & yacc
- Available via function call for all tools
- Select atoms via atom metadata
- Keywords bound to atom properties
  - `id, name, resname, resid, segid`
- Special keywords
  - `all, none, hydrogen`
- Substring and pattern matching via regular expression operator `=~`
- Number extraction operator `->`
- Complex selections can be stored in a file and used via shell substitution
- Convenience functions in `AtomicGroup` reduce selection complexity: `splitByResidue()`, `splitByMolecule()`, ...
- Selection is a copy of shared atoms
- Selection can occur at any time
- Same syntax on command line and inside code

Select non-hydrogen atoms

```
!hydrogen
```

Select CA atoms

```
name == "CA"
```

Select backbone atoms

```
name =~ "^(C|O|N|CA)$"
```

Select heavy atoms from a range of residues

```
(resid >= 10 && resid <= 20) && !hydrogen
```

Developing with LOOS

General

- Flat object hierarchy
  - `AtomicGroup` can be a residue, chain, molecule, system, …
- Built-in functions recover hierarchy:
  - `splitByMolecule()`, …
- Factory functions read models and trajectories
- Write out a PDB by printing it
- `TrajectoryWriter` writes DCD or XTC formats
- ASCII Matrix I/O compatible with gnuplot and MATLAB/Octave
- Typical analysis idiom defines system and loops over trajectory
  - Trajectory class is an iterator
  - Updating system updates all copies (selected atoms)

Python-Specific

- Core library available in Python
- Support for shallow and deep copies
- Container classes are iterable
- STL containers explicitly wrapped
- C++ exceptions translated to Python
- Use NumPy rather than LOOS for linear algebra/matrices
- Direct NumPy support
- `pyloos`
  - All Python code
  - Layered on top of C++/Swig
  - More Python-like interface
- `pyloos` includes:
  - Trajectory object/iterator
  - Virtual trajectories (many trajectories appear as a single one)
  - Optimally aligned virtual traj's
- Most tool development in Python

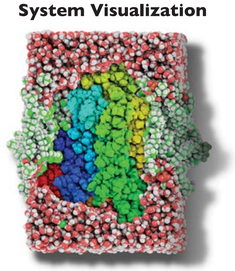
Benchmarks

Task	Language	System		
		LfB	Rhodopsin	GPCR-Complex
Iteratively Align Structures	C++	3s	35s	125s
	Python	3s	13s	130s
Interatomic distance	C++	2s	16s	120s
	Python	3s	24s	141s
All-to-all RMSD	C++	33s	95s	778s
	C++ (8-threads)	15s	33s	284s
	Python	97s	404s	4362s
Trajectory Size (GB)		0.15	2.6	21
System Size (Atoms)		2,847	46,433	196,420
Number of Frames		4,285	4,902	9,338
Selection Size (Atoms)		24	326	1,395

Intel i7-3770 @ 3.4 GHz, 32 GB RAM

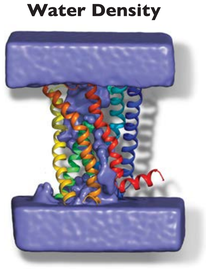
Made with LOOS

System Visualization



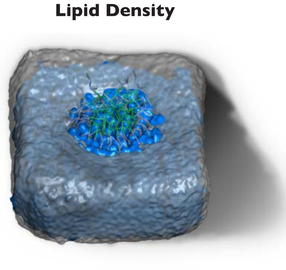
Tools: custom software

Water Density



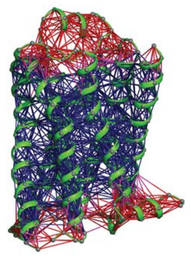
water-hist

Lipid Density



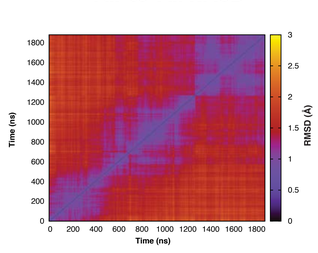
water-hist

Elastic Network Models



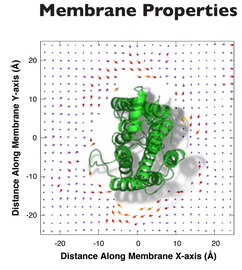
ANM/VSA (rebond)

All-to-All RMSD



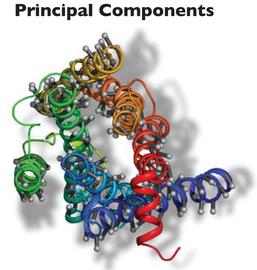
rmsds

Membrane Properties



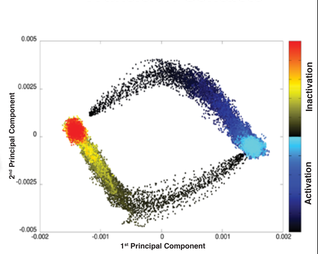
membrane_map

Principal Components



svd & porcupine

Transition Contacts



transitions_contacts