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

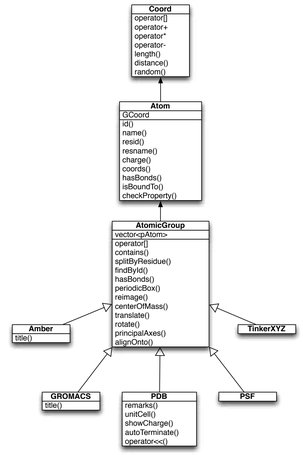
We have developed LOOS (Lightweight Object Oriented Structure-analysis) as a tool for making new tools to analyze molecular simulations. LOOS is an object-oriented library designed to facilitate the rapid development of new methods for structural analysis. LOOS includes over 140 pre-built tools for common structural analysis tasks. LOOS supports reading the native file formats of most common simulation packages and can write NAMD formats (PDB and DCD) and GROMACS XTC trajectories. A dynamic atom selection language, based on the C expression syntax, is included as part of the library and is easily accessible to both the programmer and the end user. LOOS is written in C++ and makes extensive use of Boost and the Standard Template Library. Through modern C++ design, LOOS is both simple to use (requiring knowledge of only 4 core classes and a few utility functions) and easily extensible. A Python interface to the core components of LOOS is also available, further facilitating rapid development of analysis tools and broadening the LOOS community by making it accessible to those who would otherwise be deterred by using C++. LOOS also includes a set of libraries and tools for performing elastic network model calculations that are easily extended to accommodate new methods.

Design Goals

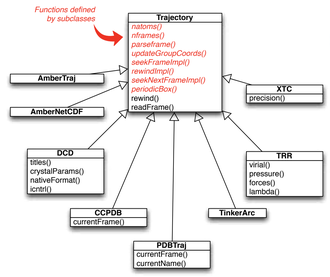
- **Lightweight**
  - Tool developers only need 4 core classes: `Coord`, `Atom`, `AtomicGroup`, `Trajectory`
  - Few external dependencies: Boost, scons, atlas/lapack *SWIG & NetCDF (optional)*
  - All available via package system on Linux
- **Extensible**
  - Polymorphic classes
  - Algorithm encapsulation
  - 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)



Trajectories



Example Code

Tracking Motion of Two Protein Segments

Read Command Line

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

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.createTrajectory(traj_file, system)
ptraj = loos.PyTraj(traj, system, skip = frames_to_skip)
```

Select "Domains"

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

Loop Over Trajectory

```
for frame in ptraj:
```

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 ptraj.currentIndex(), distance, angle, tors
```

Bundled Tools

Over 140 tools total, including 5 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 2H 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
Density Package	
blobid	Segment a density grid and find non-contiguous blobs of density
grid2xplor	Convert density grid to X-plor electron density map format for visualization
gridmask	Apply a binary mask to a density grid
near_blobs	Find residues that are near a blob for a trajectory
water-hist	Density histogram for atoms in a trajectory
Elastic Network Models	
anm	Anistropic Network Model
enmovie	Visualize ENM motions by generating a trajectory for an ENM solution
vsa	Vibrational Subsystem Analysis
Hydrogen Bonds	
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	
OptimalMembraneGenerator.py	Build membranes de novo with multiple lipid types, protein, cholesterol, asymmetric bilayers, etc

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 (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 hierarchy
  - `AtomicGroup` can be a residue, chain, molecule, system, …
- Built-in functions recover hierarchy:
  - `splitByMolecule()`, …
- Use factory functions to read models and trajectories
- Write out a PDB by printing it
- `TrajectoryWriter` can use 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
- `PyTraj` wraps a `Trajectory`
  - Use for-loop to process trajectory
  - Skip the first n-frames
  - Stride through trajectory
- `PyAlignedTraj` wraps optimally aligned virtual trajectory
  - Iterative alignment procedure
  - Align desired subset of atoms
  - Apply alignment transformation to frame (or subset)

Benchmarks

Task	Language	System: LfB	System: Opsin	System: GPCR-Complex
Iteratively Align Structures	C++	3s	58s	276s
	Python	20s	80s	394s
All-to-all RMSD	C++	91s	170s	484s
	Python	148s	233s	534s
Inter-atomic Distance	C++	0s	6s	37s
	Python	0s	6s	37s
Trajectory Size (GB)		0.15	2.25	12.19
System Size (Atoms)		2,746	47,210	276,122
Number of Frames		4,285	4,058	3,678
Selection Size (Atoms)		24	205	1,076

Intel i7-3770 @ 3.4 GHz, 32 GB RAM

Made with LOOS