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

**Tod D. Romo and Alan Grossfield**

University of Rochester Medical School, Rochester, NY, USA

http://loos.sourceforge.net

---

**Abstract**

We have developed LOOS (Lightweight Object Oriented Structure-analysis) as a tool for making new tools for analyzing molecular simulations. LOOS is an object-oriented library designed to facilitate the rapid development of new methods for structural analysis. LOOS includes over 120 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). 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 also 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.
- **Extensible**: Polymorphic classes - Algorithm encapsulation - Design patterns for easy extension
- **Easy to use**: Complex tools with minimal code - Python interface to core library - Rapid development of new tools - Templates for writing new tools for common cases - Tools are self-documented (embedded detailed help)
- **Multiplatform Support**: Linux - Mac OS X - Windows (cygwin)
- **Multipackage Support**: CHARMM/NAMD - Amber (including NetCDF) - GROMACS/MARTINI - Trinker - Easy to extend

**Class Structures**

- 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 operators - Number extraction operator -> Complex selections can be stored in a file and used via shell substitution - Convenience functions in AtomicGroup reduce selection complexity: splitByResidue(), splitByEmailolecule()...

**Example Code**

```cpp
int main() {
  #include <loos.hpp>
  from loos import *
  for j in range(max_iterations):
    model = createSystem(model_name)
    traj = createTrajectory(traj_name, model)
    AtomicGroup subset = selectAtoms(model, selection)
    pTraj traj = createTrajectory(traj_name, model)
    avg_rmsd = 0
    iterativeAlignment(ensemble)
    readTrajectory(ensemble, subset, traj)
    subset = selectAtoms(model, selection)
    # Align frame onto the optimal average
    structure.alignOnto(average)
    # Find optimal alignment
    subset = selectAtoms(model, selection)
    # Read in model and setup trajectory interface
    pTraj traj = createTrajectory(traj_name, model)
    avg_rmsd = 0
    iterativeAlignment(ensemble)
    readTrajectory(ensemble, subset, traj)
    for j in range(max_iterations):
      structure = subset[0]
      # Loop over all stored frames of the trajectory
      avg_rmsd = 0
      subset = selectAtoms(model, selection)
      for j in range(max_iterations):
        structure = subset[0]
        # Loop over all stored frames of the trajectory
        avg_rmsd = 0
        subset = selectAtoms(model, selection)
        # Align frame onto the optimal average
        structure.alignOnto(average)
        # Find optimal alignment
        subset = selectAtoms(model, selection)
        # Read in model and setup trajectory interface
        pTraj traj = createTrajectory(traj_name, model)
      # Align frame onto the optimal average
      structure.alignOnto(average)
      # Find optimal alignment
      subset = selectAtoms(model, selection)
      # Read in model and setup trajectory interface
      pTraj traj = createTrajectory(traj_name, model)
  }
  return(1)
}
```

**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 operators - Number extraction operator -> Complex selections can be stored in a file and used via shell substitution - Convenience functions in AtomicGroup reduce selection complexity: splitByResidue(), splitByEmailolecule()...

**Bundled Tools**

- Over 120 tools total, including 4 packages and 60 core tools
- Many useful member functions - Parser built using standard Unix tools - Rich atom selection language
- Based upon C/C++ expressions - SWIG (optional)
- Extensive use of Boost and the Standard Template Library.

**System Visualization**

- Water Density
- Lipid Density
- Elastic Network Models
- Simulation Convergence
  - All-to-All RMSD
  - Block Covariance Overlap Method
- Principal Components
- Peptide Location in Membrane

**Tools**

- TinkerXYZ
- LfB6 POPC
- LfB6 POPE:POPG
- Randomly orient peptides
- Place N-1 peptides
- Recurse:
  - no
  - yes
- Continue:
  - no
  - yes

**Example Code**

```cpp
# Align frame onto the optimal average
structure.alignOnto(average)
# Find optimal alignment
subset = selectAtoms(model, selection)
# Read in model and setup trajectory interface
pTraj traj = createTrajectory(traj_name, model)
```